Entender lo que armás

La teoría detrás de cada nivel del laboratorio, explicada paso a paso. Diez grupos y cincuenta niveles: de cerrar un circuito a programar una casa inteligente.

REALIZADO POR

José Gastón Cirvini

Mendoza, Argentina

DONACIONES

¡Apoyame para más proyectos! Este trabajo lo hice con mucho cariño y voy a ir mejorándolo mes a mes, así tienen una experiencia mucho mejor. Todo gratis, por el aprendizaje. Tu ayuda suma, y mucho, para mí. ¡¡Gracias!!

Invitar un cafecito (Mercado Pago, Argentina)

`cafecito.app/gastoncirvini`

Donar con PayPal (exterior)

`paypal.me/gastoncirvini`

CONCEPTO

La idea central del nivel, en palabras simples. Tiene el color del grupo.

EJEMPLO

Un circuito o un código parecido al del nivel, pero distinto: para entender, no para copiar.

CUIDADO

Los errores que casi todos cometemos la primera vez.

© José Gastón Cirvini

PROBÁ ADEMÁS

Un desafío corto para hacer en el modo libre cuando el nivel ya salió.

EN LA PLACA REAL

Lo que cambia cuando pasás del simulador a una placa de verdad.

ÍNDICE

Grupo 1 · La electricidad se mueve

1.1 Encendé un LED
1.2 Los rieles de energía
1.3 El LED tiene dirección
1.4 La resistencia justa
1.5 Dos LED a la vez

Grupo 2 · Botones y llaves

2.1 Un pulsador
2.2 Un interruptor
2.3 Las dos a la vez
2.4 Cualquiera de las dos
2.5 Timbre sin programa

Grupo 3 · Mi primer programa

3.1 Parpadeo
3.2 Más rápido
3.3 Cambio de pin
3.4 Dos LED que se turnan
3.5 Semáforo

Grupo 4 · El Arduino escucha

4.1 El Arduino lee un botón
4.2 INPUT_PULLUP
4.3 Decidir con if
4.4 Verde o rojo
4.5 Un botón con memoria

Grupo 5 · Valores que cambian

5.1 Leer un valor
5.2 Brillo regulable
5.3 La función map
5.4 Un umbral
5.5 Barra de nivel

Grupo 6 · Sensores del mundo real

6.1 Luz automática
6.2 Alarma de calor
6.3 Detector de movimiento
6.4 Sensor de estacionamiento
6.5 Temperatura y humedad

Grupo 7 · Cosas que se mueven y suenan

7.1 Una melodía
7.2 Servo a mano
7.3 Ventilador automático

© José Gastón Cirvini

7.4 Riego automático
7.5 Mezclar colores

Grupo 8 · Programar en serio

8.1 Constantes con nombre
8.2 Repetir con for
8.3 Arreglos: ida y vuelta
8.4 Funciones propias
8.5 Dos cosas a la vez

Grupo 9 · Pantallas

9.1 Hablar por el Monitor serie
9.2 Contador de 7 segmentos
9.3 Reloj de 4 dígitos
9.4 Tu nombre en la pantalla LCD
9.5 Gráficos en la pantalla OLED

Grupo 10 · Proyectos completos

10.1 Alarma con llave
10.2 Estación del clima
10.3 Maceta inteligente
10.4 Estacionamiento con luces
10.5 Proyecto final: casa inteligente

Glosario

© José Gastón Cirvini

GRUPO 1 · ETAPA: ELECTRICIDAD

La electricidad se mueve

Antes de programar hay que entender qué pasa dentro de los cables. En este grupo el Arduino no piensa: solo entrega energía, como una pila. Todo lo que armes acá lo vas a volver a usar en cada proyecto.

- Qué es un circuito cerrado y por qué sin él no pasa nada.
- Cómo está conectada por dentro la protoboard.
- Por qué un LED tiene un sentido y siempre lleva resistencia.
- Qué significa conectar dos cosas en paralelo.

1.1 Encendé un LED

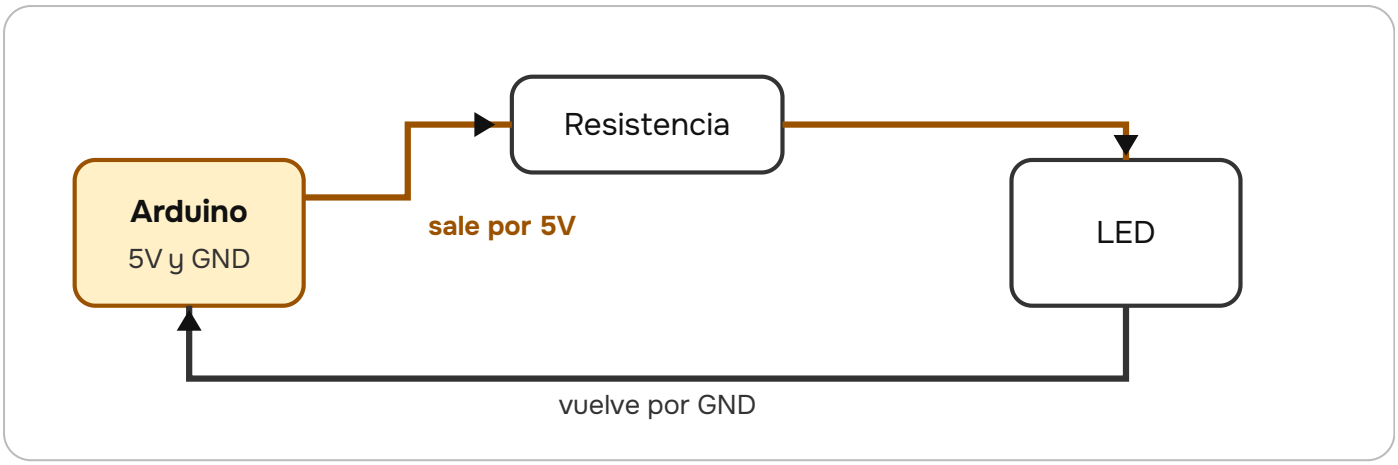
META Explicar por qué un circuito tiene que estar cerrado.

CONCEPTO

La electricidad es un movimiento de cargas diminutas por dentro de un cable. Para que se muevan necesitan dos cosas: algo que las empuje y un camino que las lleve de vuelta al punto de partida. En el Arduino, el empuje sale del pin `5V` y el regreso es el pin `GND` (tierra).

Ese camino completo, de ida y de vuelta, se llama **circuito cerrado**. Si lo cortás en cualquier punto, las cargas dejan de moverse en todo el circuito al mismo tiempo, no solo después del corte.

© José Gastón Cirvini



La corriente sale por 5V, pasa por la resistencia y el LED, y vuelve por GND. Si falta un tramo, no circula nada.

CONCEPTO · LA PROTOBOARD

La protoboard es una placa llena de agujeros que permite conectar componentes sin soldar. Por dentro tiene tiritas de metal. En la zona central, los cinco agujeros de cada columna están unidos entre sí; las columnas vecinas no se tocan. Por eso, para unir dos patas alcanza con meterlas en la misma columna.

CUIDADO

- Poner las dos patas de un mismo componente en la misma columna: quedan unidas entre sí y la corriente lo saltea.
- Olvidar el cable a GND: el camino queda abierto.
- Conectar el LED sin resistencia: recibe demasiada corriente y se quema.

PROBÁ ADEMÁS

Con el LED encendido, sacá un cable cualquiera y volvé a ponerlo. Después cambiá de lugar la resistencia: ¿importa si va antes o después del LED?

EN LA PLACA REAL

Los cables y las patas tienen que entrar hasta el fondo. Un falso contacto es la falla más común y la más difícil de ver. Antes de enchufar el Arduino a la computadora, revisá que 5V y GND no se toquen.

Palabras nuevas: **circuito cerrado** **5V** **GND** **protoboard**

© José Gastón Cirvini

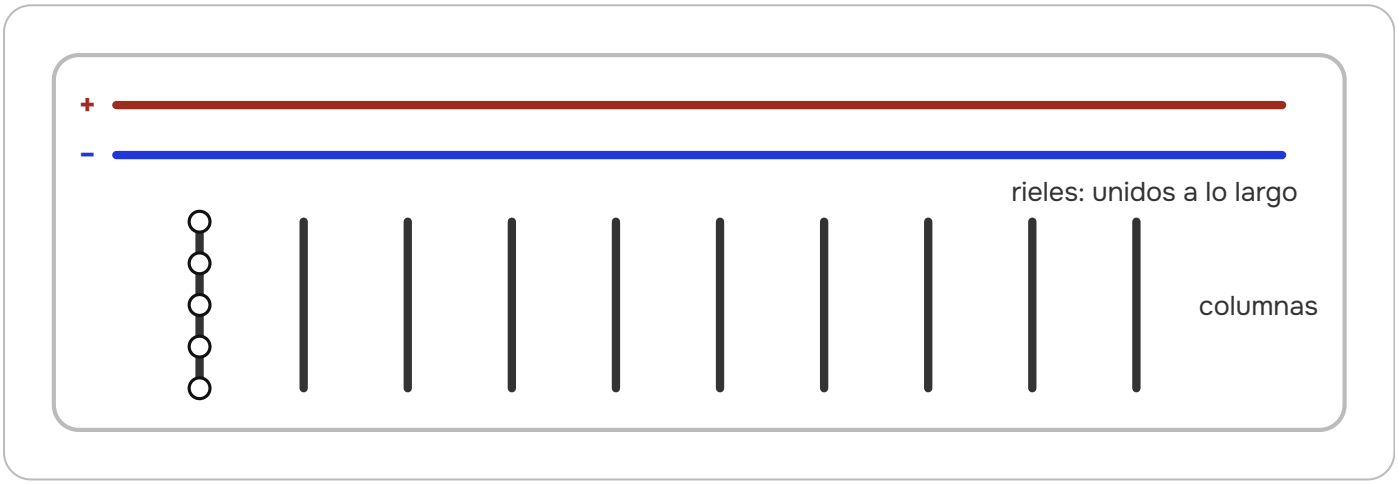
1.2 Los rieles de energía

META Repartir energía a muchos componentes con solo dos cables.

CONCEPTO

El Arduino tiene un solo pin de 5V y pocos de GND. Un proyecto real necesita alimentar varios componentes a la vez. La solución está en los bordes de la protoboard: dos filas largas llamadas **rieles**, marcadas con una línea roja (+) y una azul (-).

A diferencia de las columnas del centro, cada riel está unido de punta a punta. Si llevás 5V a un agujero del riel rojo, todo el riel queda con 5V. Lo mismo con GND en el azul.



Rieles: una sola tira a lo largo. Columnas: cinco agujeros unidos a lo alto, separados de las columnas vecinas.

CUIDADO

- Cruzar los rieles: 5V al azul y GND al rojo funciona igual, pero confunde y lleva a errores.
- Unir el riel + con el riel - con un cable: eso es un cortocircuito.
- Creer que el riel de arriba y el de abajo están conectados: son independientes, hay que unirlos con cables.

PROBÁ ADEMÁS

Alimentá también los rieles de abajo usando dos cables puente, y encendé un LED desde ahí.

Palabras nuevas: **riel** **cortocircuito** **alimentar**

© José Gastón Cirvini

1.3 El LED tiene dirección

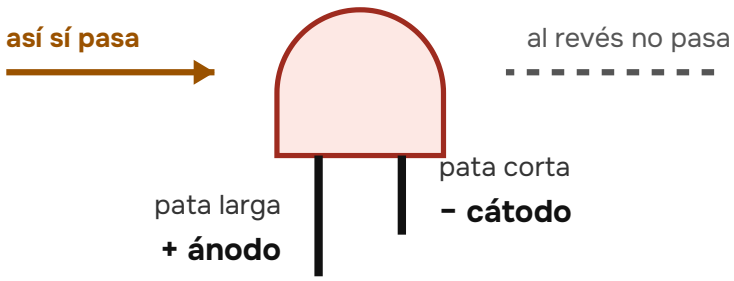
META Reconocer la pata positiva y la negativa de un componente.

> Hasta acá los LED estaban siempre bien puestos. ¿Qué pasa si uno queda al revés?

CONCEPTO

LED significa «diodo emisor de luz». Un diodo es una válvula: deja pasar la corriente en un solo sentido. Los componentes que tienen un sentido obligatorio se llaman **polarizados**.

La pata positiva se llama **ánodo** y es la más larga. La negativa se llama **cátodo**, es más corta y suele coincidir con un borde plano en la cabeza del LED. La corriente entra por el ánodo y sale por el cátodo.



El ánodo mira hacia el positivo; el cátodo, hacia GND.

CONCEPTO · LEER EL ERROR

Un LED al revés no se rompe: simplemente no enciende. Aprender a buscar un error con método vale más que evitarlo. Seguí el camino con el dedo desde 5V hasta GND y preguntate en cada tramo: ¿por acá puede pasar la corriente?

CUIDADO

- No todos los componentes son polarizados: una resistencia se puede poner en cualquier sentido.
- El zumbador también tiene + y –, igual que el LED.

© José Gastón Cirvini

PROBÁ ADEMÁS

Armá un circuito que funcione y pedile a un compañero que cambie un solo cable sin que mires. Encontrá el cambio usando el Asistente.

EN LA PLACA REAL

En el simulador ves las marcas + y −. Un LED real no las trae: se reconoce por la pata larga (+) y por el borde plano de la cabeza (−). Si ya le cortaste las patas, mirá adentro: la pieza de metal más grande es el cátodo.

Palabras nuevas: **diodo** **ánodo** **cátodo** **polarizado**

1.4 La resistencia justa

META Elegir el valor de una resistencia según para qué la uses.

> Ya sabés que el LED necesita una resistencia. Falta ver cuál.

CONCEPTO

Pensá en una manguera. El **voltaje** (volts, V) es la presión del agua. La **corriente** (amperes, A) es cuánta agua pasa por segundo. La **resistencia** (ohm, Ω) es qué tan angosta es la manguera: cuanto más angosta, menos agua pasa.

Regla práctica: más ohm, menos corriente, menos brillo.

Y la respuesta a la pregunta del nivel 1.1: en un camino en fila la corriente es la misma en todos los puntos. Da igual si la resistencia va antes o después del LED.

PARA IR MÁS LEJOS · LA LEY DE OHM

Las tres magnitudes se relacionan así: **corriente = voltaje ÷ resistencia**. Con el mismo voltaje, el doble de resistencia deja pasar la mitad de corriente.

La cuenta para un LED: el Arduino entrega 5 V. Un LED rojo «se queda» con unos 2 V, así que a la resistencia le tocan 3 V. Con 220 Ω: 3 ÷ 220 = 0,0136 A, o sea unos 14 miliamperes. Un LED común aguanta hasta 20 mA: perfecto.

© José Gastón Cirvini

VALOR	CORRIENTE EN UN LED ROJO	QUÉ PASA	PARA QUÉ SE USA
100 Ω	30 mA	Se pasa del límite	Casi nunca con LED
220 Ω	14 mA	Brilla bien	LED con 5 V
330 Ω	9 mA	Brilla un poco menos	LED, más seguro
1 kΩ	3 mA	Brilla poco	Señales, LED indicador
10 kΩ	0,3 mA	Casi no se ve	Pull-down de un botón

CUIDADO

- «k» significa mil: 10 kΩ son 10.000 Ω. Confundir 220 Ω con 220 kΩ es un error clásico.
- Más resistencia no es «más seguro» para todo: si es demasiada, el componente no funciona.

PROBÁ ADEMÁS

Hacé la cuenta para un LED azul, que se queda con unos 3 V. ¿Qué corriente pasa con 220 Ω? ¿Brilla más o menos que el rojo?

EN LA PLACA REAL

Una resistencia real no trae el número escrito: lo indican bandas de colores. 220 Ω es rojo, rojo, marrón. 10 kΩ es marrón, negro, naranja. La última banda, dorada, indica la precisión.

Palabras nuevas: **voltaje** **corriente** **ohm (Ω)** **ley de Ohm** **miliampere**

1.5 Dos LED a la vez

META Distinguir serie de paralelo y saber por qué cada LED lleva su resistencia.

© José Gastón Cirvini

CONCEPTO

Dos componentes están en **paralelo** cuando cada uno tiene su propio camino entre el positivo y GND. La corriente se reparte: una parte va por un camino y otra por el otro. Si uno se corta, el otro sigue funcionando. Así están conectadas las luces de tu casa.

Lo contrario es la conexión en **serie**: los componentes van en fila, uno detrás del otro, sobre un mismo camino. La resistencia y el LED que venís usando están en serie.

Cada LED necesita su propia resistencia porque no hay dos LED exactamente iguales. Si comparten una sola, el que deja pasar la corriente con más facilidad se lleva casi toda y el otro queda apagado o débil.

CUIDADO

- Poner los dos LED en fila con una sola resistencia no es paralelo, es serie: se reparten el voltaje y puede no alcanzar para ninguno.
- Cada camino nuevo suma corriente. El Arduino tiene un límite: no alimentes decenas de LED directo desde sus pines.

PROBÁ ADEMÁS

Agregá un tercer LED de otro color. Después sacá uno: ¿los otros cambian de brillo?

Palabras nuevas: **paralelo** **serie**

© José Gastón Cirvini

GRUPO 2 · ETAPA: CONTROL

Botones y llaves

Un circuito que siempre está encendido sirve de poco. Acá aprendés a decidir cuándo pasa la corriente, y descubrís que la lógica de la programación ya existe en los cables.

- La diferencia entre un pulsador y un interruptor.
- Cómo dos llaves en serie forman una «Y» y en paralelo una «O».
- Qué es un actuador.

2.1 Un pulsador

META Usar bien las palabras «abierto» y «cerrado».

> En el grupo 1 los circuitos estaban siempre encendidos. Ahora vas a cortarlos y cerrarlos cuando quieras.

CONCEPTO

Un pulsador es un puente de metal con un resorte. Suelto, el puente está levantado y el circuito queda **abierto**: no pasa corriente. Apretado, el puente baja, une los dos contactos y el circuito queda **cerrado**.

Atención con las palabras: en electricidad, «cerrado» quiere decir que funciona y «abierto» que está cortado. Es al revés que una canilla.

El pulsador va en serie, como la resistencia y el LED del nivel 1.5: en la misma fila que el resto, para que toda la corriente tenga que pasar por él.

CUIDADO

- Conectar el pulsador «al costado» del LED en vez de en el camino: el LED queda siempre encendido.
- Usar los dos cables en el mismo contacto del pulsador: nunca cierra nada.

© José Gastón Cirvini

PROBÁ ADEMÁS

Mové el pulsador a otro lugar del camino: entre el LED y GND. ¿Sigue funcionando? ¿Por qué?

EN LA PLACA REAL

El pulsador más común tiene cuatro patas, unidas de a dos por dentro. Si lo conectás por el par equivocado queda siempre cerrado: giralo un cuarto de vuelta.

Palabras nuevas: **abierto** **cerrado** **contacto** **pulsador**

2.2 Un interruptor

META Decidir cuándo conviene un interruptor y cuándo un pulsador.

CONCEPTO

El interruptor hace el mismo trabajo que el pulsador, pero no tiene resorte: se queda en la posición en que lo dejás. Esa posición, abierto o cerrado, es su **estado**, y el interruptor lo conserva sin ayuda: tiene memoria mecánica.

La elección depende de lo que quieras lograr. Un timbre usa pulsador porque debe sonar solo mientras apretás. Una lámpara usa interruptor porque debe quedar encendida. Más adelante vas a ver que, con un programa, un pulsador puede comportarse como interruptor.

	PULSADOR	INTERRUPTOR
Al soltarlo	Vuelve solo a abierto	Se queda donde está
Sirve para	Acciones momentáneas	Estados que duran
Ejemplos	Timbre, teclado, joystick	Llave de luz, encendido

CONCEPTO · POR DENTRO

Un interruptor es una lámina de metal que una palanca apoya contra un contacto o aleja de él. No tiene electrónica: es puro movimiento. Por eso no gasta energía para recordar su posición.

© José Gastón Cirvini

CUIDADO

- Un interruptor no frena la corriente: el LED sigue necesitando su resistencia.
- Abierto no es roto. Antes de buscar un cable suelto, mirá en qué posición está la llave.

PROBÁ ADEMÁS

Poné un interruptor y un pulsador en serie. ¿Qué tiene que pasar para que el LED encienda?

Palabras nuevas: **interruptor** **estado**

2.3 Las dos a la vez

META Armar y reconocer una «Y» lógica con cables.

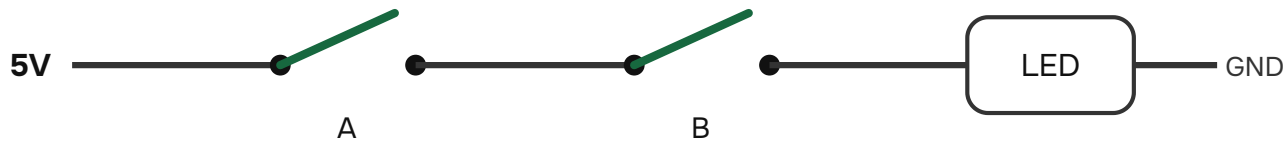
> Con un interruptor decidís vos. Con dos, empieza a decidir una regla.

CONCEPTO

Con dos interruptores en serie, la corriente tiene que atravesar los dos. Si uno solo está abierto, el camino se corta. El LED enciende únicamente cuando el primero **Y** el segundo están cerrados.

Esa regla se llama **Y lógico** (en inglés, AND). Una tabla que muestra todos los casos posibles se llama **tabla de verdad**.

En un programa, esta misma regla se escribe con el símbolo `&&`. Lo vas a usar en el grupo 4.



En serie: un solo camino que pasa por A y después por B.

© José Gastón Cirvini

A	B	LED (A Y B)
abierto	abierto	apagado
cerrado	abierto	apagado
abierto	cerrado	apagado
cerrado	cerrado	**encendido**

EJEMPLO · DÓNDE SE USA

Una prensa industrial que solo baja si el operario aprieta dos botones, uno con cada mano: así se asegura de que las manos estén lejos. Un microondas que solo arranca si la puerta está cerrada Y se apretó Inicio.

PROBÁ ADEMÁS

Agregá un tercer interruptor en serie. ¿Cuántas filas tendría ahora la tabla de verdad?

Palabras nuevas: **Y lógico (AND)** **tabla de verdad**

2.4 Cualquiera de las dos

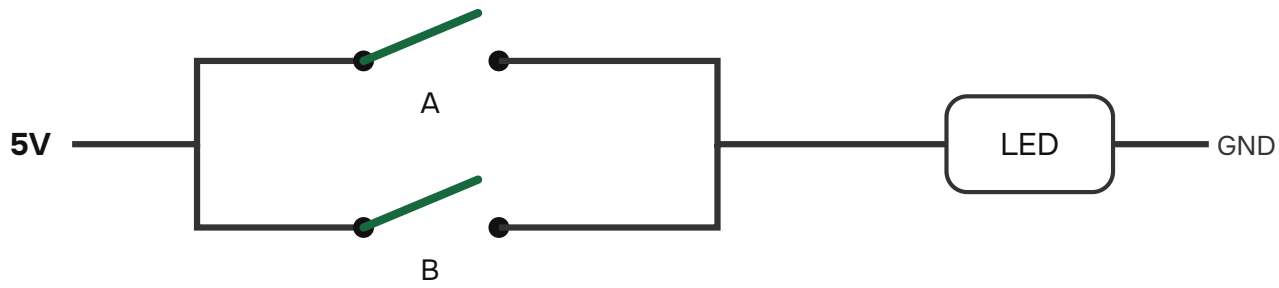
META Armar una «O» lógica y diferenciarla de la «Y».

> En el nivel anterior los interruptores iban en fila. Mirá qué cambia cuando cada uno tiene su camino.

CONCEPTO

Si cada interruptor tiene su propio camino y los dos caminos se juntan antes del LED, alcanza con que uno esté cerrado. El LED enciende cuando el primero **O** el segundo están cerrados (o los dos). Es el **O lógico** (OR).

Regla para recordar: **serie = Y**, **paralelo = O**. En un programa, la O se escribe `||`.



En paralelo: dos caminos que se juntan. Cualquiera de los dos alcanza.

A	B	LED (A O B)
abierto	abierto	apagado
cerrado	abierto	**encendido**
abierto	cerrado	**encendido**
cerrado	cerrado	**encendido**

CUIDADO

En el habla diaria «o» a veces significa «uno u otro, pero no los dos». En lógica, el O incluye el caso en que los dos son verdaderos.

PROBÁ ADEMÁS

Combiná las dos ideas: el LED enciende si está cerrado A, y además está cerrado B o C. ¿Cuál va en serie y cuáles en paralelo?

Palabras nuevas: **O lógico (OR)**

2.5 Timbre sin programa

META Cambiar un actuador por otro sin rehacer el circuito.

> Hasta ahora la salida fue siempre un LED. El mismo circuito sirve para otras cosas.

© José Gastón Cirvini

CONCEPTO

Todo sistema electrónico tiene dos tipos de piezas. Las **entradas** le avisan qué pasa afuera: un botón, un sensor. Las **salidas** o **actuadores** hacen algo en el mundo: dan luz, suenan, se mueven.

El zumbador es un actuador que convierte la corriente en sonido. En el circuito ocupa el mismo lugar que el LED: por eso podés reemplazar uno por otro sin cambiar el resto. Esa idea, piezas intercambiables que cumplen un mismo papel, es la base de todo diseño.

CUIDADO

- El zumbador es polarizado: tiene + y −.
- No necesita resistencia: ya limita la corriente por sí mismo.

PROBÁ ADEMÁS

Hacé que al apretar suene el zumbador y además encienda un LED. ¿Los conectás en serie o en paralelo?

Palabras nuevas: **entrada** **salida** **actuador**

© José Gastón Cirvini

GRUPO 3 · ETAPA: PROGRAMACIÓN

Mi primer programa

Hasta acá el Arduino fue una pila cara. Ahora empieza a decidir. Las reglas que armaste con interruptores, la Y y la O, van a volver escritas como código. Un programa es una lista de órdenes escritas en un idioma muy estricto, que la placa cumple al pie de la letra.

- Las dos partes de todo programa: `setup` y `loop`.
- Cómo encender y apagar un pin, y cómo esperar.
- Las reglas de escritura: punto y coma, llaves, mayúsculas.
- Cómo pensar una secuencia.

3.1 Parpadeo

META **Leer un programa de Arduino y decir qué hace cada línea.**

> En los grupos 1 y 2 decidías vos, con la mano, cuándo pasaba la corriente. Ahora lo va a decidir un programa.

CONCEPTO

Dentro del Arduino hay un **microcontrolador**: una computadora diminuta. Ejecuta un programa instrucción por instrucción, millones por segundo.

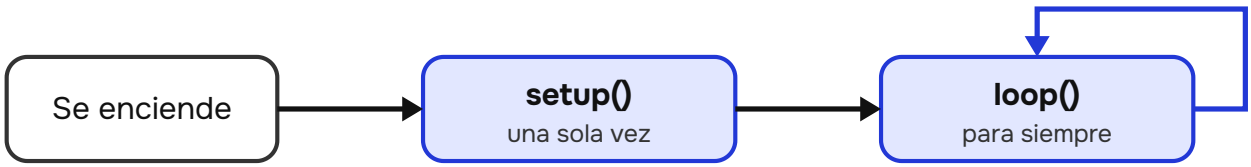
Los **pines digitales** (del 0 al 13) son contactos que el programa puede poner en 5 V o en 0 V. Un pin en 5 V funciona igual que el pin 5V que ya usaste. La diferencia: ahora lo manejás vos.

CONCEPTO · LAS DOS PARTES DE UN PROGRAMA

`setup()` se ejecuta una sola vez, al encender. Sirve para preparar.

`loop()` se repite sin fin. Ahí va lo que el Arduino hace todo el tiempo.

© José Gastón Cirvini



El programa nunca «termina»: al llegar al final de loop, vuelve a empezarlo.

EJEMPLO · UN LED EN EL PIN 7

```
void setup() {
  pinMode(7, OUTPUT);     // el pin 7 va a ser una salida
}

void loop() {
  digitalWrite(7, HIGH);  // pin 7 en 5 V: LED encendido
  delay(2000);            // esperar 2 segundos
  digitalWrite(7, LOW);   // pin 7 en 0 V: LED apagado
  delay(500);             // esperar medio segundo
}
```

CONCEPTO · LAS REGLAS DE ESCRITURA

- Cada instrucción termina con **punto y coma** `;`.
- Las **llaves** `{ }` marcan dónde empieza y termina un bloque.
- Las **mayúsculas importan**: `digitalWrite` funciona, `digitalwrite` no.
- Lo que sigue a `//` es un **comentario**: una nota para personas, que el Arduino ignora.

CUIDADO

- Olvidar `pinMode`: en una placa real el pin casi no entrega corriente y el LED queda muy tenue o apagado.
- Borrar una llave sin querer: el programa no se puede cargar y el error suele aparecer lejos de donde está el problema.

PROBÁ ADEMÁS

En el modo libre, escribí el parpadeo de memoria, sin mirar. Usá «Verificar» para que el editor te marque lo que falte.

© José Gastón Cirvini

EN LA PLACA REAL

La placa trae un LED propio, marcado con la letra L y conectado al pin 13. Por eso el parpadeo en el pin 13 se puede probar sin conectar nada.

Palabras nuevas: **microcontrolador** **pin digital** **setup** **loop** **HIGH / LOW** **comentario**

3.2 Más rápido

META Controlar el tiempo de un programa y dominar el ciclo editar, cargar y probar.

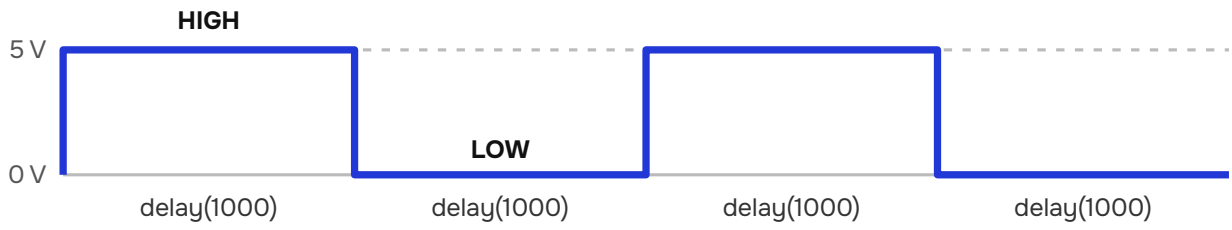
> El parpadeo del nivel anterior tenía un ritmo fijo. Ese ritmo está escrito en el código y se puede cambiar.

CONCEPTO

`delay(1000)` detiene el programa durante 1000 **milisegundos**. Un milisegundo es la milésima parte de un segundo, así que 1000 ms son 1 segundo, 500 ms medio segundo y 100 ms una décima.

El número entre paréntesis se llama **argumento**: es el dato que le pasás a la instrucción para que sepa cuánto esperar.

Cambiar el texto del programa no cambia nada en la placa. Hay que **cargarlo**. Al cargar pasan dos cosas: primero el código se traduce al idioma de la placa (eso se llama **compilar**, y es el momento en que aparecen los errores de escritura) y después se envía al Arduino, que borra el programa anterior y arranca el nuevo.



Cada delay es un tramo de la línea de tiempo. Achicar el número acorta el tramo.

© José Gastón Cirvini

CUIDADO

- Cambiar un solo `delay`: el LED queda mucho tiempo encendido y poco apagado (o al revés).
- Con delays menores a unos 15 ms el ojo ya no ve el parpadeo: parece un LED encendido más tenue.
- Editar y olvidarse de cargar: el Arduino sigue con el programa viejo.

PROBÁ ADEMÁS

Hacé un latido: dos destellos cortos seguidos y una pausa larga. ¿Cuántos `digitalWrite` y cuántos `delay` necesitás?

EN LA PLACA REAL

En el laboratorio cargás con un botón. Con una placa real se usa el programa Arduino IDE: conectás el cable USB, elegís el modelo de placa y el puerto, y tocás la flecha de Subir.

Palabras nuevas: **milisegundo** **argumento** **cargar** **compilar**

3.3 Cambio de pin

META Hacer coincidir lo que dice el código con lo que hay en la mesa.

> Ya cambiaste el tiempo. Ahora vas a cambiar el lugar: el pin.

CONCEPTO

Un proyecto de Arduino tiene dos mitades: el **hardware** (lo que se toca: placa, cables, componentes) y el **software** (el programa). Ninguna sabe nada de la otra. El único puente entre las dos es el **número de pin**.

Si el programa dice 8 y el cable está en el 13, no hay error ni aviso: el pin 8 se enciende y se apaga perfectamente, solo que ahí no hay nada conectado. Por eso, cuando algo no funciona, hay que revisar las dos mitades.

© José Gastón Cirvini

EJEMPLO · UN MÉTODO PARA REVISAR

- En el código: ¿qué número aparece en `pinMode`? ¿Es el mismo en todos los `digitalWrite`?
- En la mesa: ¿a qué pin llega el cable? Seguilo con el dedo.
- ¿Coinciden? Si no, cambiá una de las dos cosas, no las dos.

CUIDADO

- Cambiar el número en `digitalWrite` y olvidarlo en `pinMode`.
- Los pines 0 y 1 se usan para comunicarse con la computadora: mejor evitarlos.

PROBÁ ADEMÁS

Reescribí el programa para que el número de pin aparezca una sola vez. Una pista: se puede guardar en una caja con nombre. Lo vas a ver en detalle en el grupo 8.

Palabras nuevas: **hardware** **software**

3.4 Dos LED que se turnan

META Seguir un programa línea por línea y anticipar qué va a pasar.

> Con un solo LED, el orden de las instrucciones casi no se notaba. Con dos, el orden lo es todo.

CONCEPTO

El Arduino ejecuta las instrucciones **en orden**, de arriba hacia abajo, de a una. Nunca hace dos cosas a la vez. Lo que parece simultáneo son instrucciones seguidas sin espera en el medio: tarda millonésimas de segundo en pasar de una a otra.

Un pin **recuerda** su último valor. Si lo ponés en HIGH, sigue en HIGH hasta que otra instrucción lo cambie, aunque pasen diez `delay`.

EJEMPLO · SEGUIR EL PROGRAMA CON UNA TABLA

Anotar el estado de cada salida después de cada línea se llama hacer una **prueba de escritorio**. Es la mejor herramienta para entender un código ajeno.

```
digitalWrite(5, HIGH);
digitalWrite(6, HIGH);
delay(300);
digitalWrite(5, LOW);
delay(300);
digitalWrite(6, LOW);
delay(300);
```

DESPUÉS DE LA LÍNEA	PIN 5	PIN 6
1 y 2	encendido	encendido
4	apagado	encendido
6	apagado	apagado

CUIDADO

Olvidar apagar. Si encendés el segundo LED sin apagar el primero, quedan los dos encendidos: el pin no se apaga solo.

PROBÁ ADEMÁS

Hacé una prueba de escritorio del código del nivel antes de cargarlo. Después comprobá si acertaste.

Palabras nuevas: **secuencia** **prueba de escritorio**

3.5 Semáforo

META Pasar de una idea en palabras a un programa.

> Este nivel junta todo el grupo: varios pines, varios tiempos y un orden que respetar.

© José Gastón Cirvini

CONCEPTO

Un programa empieza antes de tocar el teclado. Primero se escribe, en palabras comunes, qué tiene que pasar y en qué orden. Esa lista de pasos se llama **algoritmo**. Recién después se traduce cada paso a instrucciones.

PASO DEL ALGORITMO	INSTRUCCIONES
Encender el verde (pin 8) y esperar	`digitalWrite(8, HIGH); delay(...);`
Apagar el verde	`digitalWrite(8, LOW);`
Encender el amarillo (pin 9) y esperar poco	`digitalWrite(9, HIGH); delay(...);`
Apagar el amarillo	`digitalWrite(9, LOW);`
Encender el rojo (pin 10) y esperar	`digitalWrite(10, HIGH); delay(...);`
Apagar el rojo y volver a empezar	`digitalWrite(10, LOW);` y termina `loop`

CONCEPTO · UN PROYECTO SE ARMA POR PARTES

Tres LED, tres resistencias y seis cables son muchos lugares donde equivocarse. Conectá un LED, comprobá que enciende en su momento y recién ahí sumá el siguiente. Si algo falla, sabés que el problema está en lo último que agregaste.

CUIDADO

- Cruzar dos cables entre pines: el semáforo funciona, pero con los colores en otro orden.
- Compartir una sola resistencia entre los tres LED: como nunca encienden juntos, acá funciona, pero es una mala costumbre.

PROBÁ ADEMÁS

Escribí primero el algoritmo y después el código de un semáforo peatonal: verde fijo, verde que parpadea tres veces y rojo.

Palabras nuevas: **algoritmo**

© José Gastón Cirvini

GRUPO 4 · ETAPA: ENTRADAS

El Arduino escucha

Hasta acá el Arduino solo daba órdenes. Ahora también recibe información: lee un botón y decide qué hacer. Con entradas y decisiones, un programa deja de ser una secuencia fija.

- Cómo se lee un pin y por qué no puede quedar «en el aire».
- Las resistencias pull-down y pull-up.
- Decidir con `if` y `else`.
- Variables: la memoria del programa.

4.1 El Arduino lee un botón

META Leer un pulsador desde el programa sin lecturas falsas.

> En el grupo 2 el pulsador cortaba la corriente del LED. Ahora no toca al LED: le avisa al Arduino, y el Arduino decide.

CONCEPTO

Un pin puede trabajar como **entrada**. Con `pinMode(2, INPUT)` deja de entregar corriente y se pone a medir. `digitalRead(2)` devuelve `HIGH` si al pin le llegan 5 V y `LOW` si está en 0 V.

CONCEPTO · EL PIN FLOTANTE

Con el pulsador apretado, al pin le llegan 5 V. ¿Y suelto? No queda conectado a nada. Un pin así se llama **flotante** y lee cualquier cosa: funciona como una antena que capta la electricidad del ambiente.

La solución es darle un valor de reposo. Una resistencia de 10 kΩ entre el pin y GND lo mantiene en 0 V mientras nadie aprieta. Se llama **pull-down** porque «tira hacia abajo». Es grande a propósito: cuando apretás, casi toda la corriente va al pin y casi nada se pierde hacia GND.

© José Gastón Cirvini

EJEMPLO · UN BOTÓN EN EL PIN 4 QUE APAGA UN LED

```
void setup() {
  pinMode(4, INPUT);
  pinMode(9, OUTPUT);
}

void loop() {
  int lectura = digitalRead(4);    // HIGH o LOW
  digitalWrite(9, !lectura);       // el LED hace lo contrario
}
```

CUIDADO

- Olvidar la resistencia pull-down: el LED titila solo o responde cuando acercás la mano.
- Usar 220 Ω como pull-down: funciona, pero desperdicia corriente cada vez que apretás.
- Poner `OUTPUT` en el pin del botón: deja de leer.

PROBÁ ADEMÁS

Sacá la resistencia pull-down y mirá el Asistente. Después volvé a ponerla.

EN LA PLACA REAL

Un pin flotante real cambia de valor con solo acercar un dedo. Es una de las fallas más desconcertantes para quien empieza.

Palabras nuevas: **entrada digital** **digitalRead** **pin flotante** **pull-down**

4.2 INPUT_PULLUP

META Conectar un botón sin resistencia externa y leerlo bien.

> La resistencia del nivel anterior es tan común que el Arduino ya la trae adentro.

CONCEPTO

`pinMode(2, INPUT_PULLUP)` activa una resistencia interna que une el pin con 5 V. Es un **pull-up**: el reposo ahora es `HIGH`. El pulsador se conecta entre el pin y GND, sin nada más.

La lógica queda **invertida**: suelto lee `HIGH`, apretado lee `LOW`. Parece raro, pero es la forma más usada porque ahorra un componente y dos cables.

	PULL-DOWN EXTERNO	INPUT_PULLUP
El pulsador va entre	el pin y 5V	el pin y GND
Resistencia	10 kΩ a GND, la ponés vos	interna, a 5V
Suelto	LOW	HIGH
Apretado	HIGH	LOW

CONCEPTO · EL SIGNO !

`!` significa «lo contrario de». `!HIGH` es `LOW`. Sirve para dar vuelta una lectura invertida y volver a pensar en positivo.

CUIDADO

- Usar `INPUT_PULLUP` y conectar el pulsador a 5V: el pin lee siempre `HIGH`.
- Olvidar que está invertido: el LED queda encendido y se apaga al apretar.

PROBÁ ADEMÁS

Escribí el programa sin usar `!`, pensando directamente en `LOW` como «apretado».

Palabras nuevas: **pull-up** **INPUT_PULLUP** **lógica invertida** **negación (!)**

4.3 Decidir con if

META Escribir una decisión con `if` y `else`.

> Hasta ahora el LED copiaba al botón con una sola línea. Para hacer cosas distintas en cada caso hace falta decidir.

© José Gastón Cirvini

CONCEPTO

`if` significa «si». Entre paréntesis va una **condición**: una pregunta que solo puede ser verdadera o falsa. Si es verdadera, se ejecuta el bloque entre llaves. `else` («si no») es opcional y se ejecuta en el caso contrario.

CONCEPTO · VARIABLES

Una **variable** es una caja con nombre donde el programa guarda un dato. `int boton = digitalRead(2);` hace tres cosas: crea la caja, le pone nombre (`boton`) y guarda adentro la lectura. `int` indica el tipo de dato: un número entero.

CONCEPTO · = NO ES ==

Un signo `=` **guarda**: «poné este valor en la caja». Dos signos `==` **preguntan**: «¿son iguales?». Dentro de un `if` casi siempre van dos.

EJEMPLO · UN SENSOR DE PUERTA

Un interruptor en el pin 6 (con pull-up) está cerrado cuando la puerta está cerrada.

```
int puerta = digitalRead(6);

if (puerta == HIGH) {        // abierta
  digitalWrite(12, HIGH);    // luz de aviso
} else {
  digitalWrite(12, LOW);
}
```

CUIDADO

- Escribir `if (boton = LOW)`: en vez de preguntar, guarda. El editor del laboratorio lo marca.
- Poner punto y coma después del paréntesis del `if`: la condición queda sin bloque.

PROBÁ ADEMÁS

Invertí el comportamiento cambiando una sola palabra del programa.

Palabras nuevas: **if** **else** **condición** **variable** **int** **==**

© José Gastón Cirvini

4.4 Verde o rojo

META Controlar varias salidas desde una misma decisión.

> Un `if` puede hacer mucho más que prender un LED.

CONCEPTO

Entre las llaves de un `if` o de un `else` entran todas las instrucciones que quieras. El conjunto se llama **bloque**. El programa ejecuta uno de los dos bloques completo y saltea el otro.

Cuando dos salidas nunca deben coincidir, cada bloque tiene que ocuparse de las dos: encender una *y* apagar la otra. Un pin no se apaga solo.

CONCEPTO · LA SANGRÍA

Las líneas dentro de un bloque se escriben corridas hacia la derecha. Al Arduino le da igual, pero a vos te muestra de un vistazo qué está adentro de qué. El editor del laboratorio la pone solo al apretar Enter.

EJEMPLO · BARRERA DE ESTACIONAMIENTO

```
if (digitalRead(3) == LOW) {   // hay un auto
  digitalWrite(10, HIGH);      // luz verde
  digitalWrite(11, LOW);
  tone(8, 900);                // aviso sonoro
} else {
  digitalWrite(10, LOW);
  digitalWrite(11, HIGH);      // luz roja
  noTone(8);
}
```

CUIDADO

- Encender en un bloque y olvidarse de apagar en el otro: quedan las dos luces prendidas.
- Perder una llave de cierre: el `else` queda «huérfano» y el programa no se carga.

© José Gastón Cirvini

PROBÁ ADEMÁS

Agregá un tercer LED que esté encendido siempre, sin importar el botón. ¿Dónde va esa línea?

Palabras nuevas: **bloque** **sangría**

4.5 Un botón con memoria

META Hacer que un pulsador se comporte como interruptor usando variables.

> En el grupo 2 la memoria la ponía el interruptor, con su palanca. Ahora la pone el programa.

CONCEPTO

`loop` se repite miles de veces por segundo. Si el programa dijera «si está apretado, cambiar el LED», lo cambiaría miles de veces mientras dura un solo toque.

Lo que interesa es el **instante** en que el botón pasa de suelto a apretado. Ese cambio se llama **flanco**. Para detectarlo hay que comparar la lectura de ahora con la de la vuelta anterior, y para eso hace falta una variable que la recuerde.

CONCEPTO · VARIABLES QUE SOBREVIVEN

Una variable creada **fuera** de `setup` y `loop` es **global**: conserva su valor entre una vuelta y la siguiente. Una creada dentro de `loop` nace y muere en cada vuelta.

`bool` es un tipo que solo guarda verdadero (`true`) o falso (`false`). Es ideal para estados de sí o no.

© José Gastón Cirvini

EJEMPLO · CONTAR TOQUES

El mismo truco sirve para contar. Acá cada toque suma uno.

```
int toques = 0;
int anterior = HIGH;

void loop() {
  int ahora = digitalRead(2);
  if (ahora == LOW && anterior == HIGH) {
    toques = toques + 1;       // un flanco, un toque
  }
  anterior = ahora;            // recordar para la próxima vuelta
  delay(20);
}
```

CUIDADO

- Olvidar `anterior = ahora;`: la comparación usa siempre un dato viejo.
- Declarar `anterior` dentro de `loop`: se borra en cada vuelta y no recuerda nada.

PROBÁ ADEMÁS

Hacé que el LED se encienda recién al tercer toque.

EN LA PLACA REAL

Un pulsador real no hace un contacto limpio: rebota durante unos milisegundos y el Arduino ve varios toques. Ese `delay(20)` está para dejar pasar el rebote.

Palabras nuevas: **flanco** **variable global** **bool** **rebote**

© José Gastón Cirvini

GRUPO 5 · ETAPA: VALORES QUE VARÍAN

Valores que cambian

El mundo no es solo sí o no. Una perilla, la luz o la temperatura pueden valer muchas cosas. En este grupo el Arduino aprende a medir cuánto y a responder con más o menos fuerza.

- Entradas analógicas y el Monitor serie.
- PWM: salidas con intensidad.
- Convertir escalas con `map`.
- Umbrales, y el primer encuentro con arreglos y `for`.

5.1 Leer un valor

META Leer una entrada analógica y ver el número en pantalla.

> Un botón tiene dos respuestas. Una perilla tiene más de mil.

CONCEPTO

Una señal **digital** tiene dos valores: 0 V o 5 V. Una señal **analógica** puede valer cualquier cosa entre medio: 1,3 V, 2,71 V, 4 V.

Los pines `A0` a `A5` miden ese voltaje. `analogRead(A0)` lo convierte en un número entero entre 0 y 1023: 0 es 0 V, 1023 es 5 V y 512 es la mitad. Son 1024 escalones porque el conversor trabaja con 10 bits.

CONCEPTO · EL MONITOR SERIE

El Arduino no tiene pantalla, pero puede enviar texto a la computadora por el mismo cable con el que se carga. `Serial.begin(9600)` abre el canal (9600 es la velocidad) y `Serial.println(valor)` escribe una línea. Es tu ventana para ver qué pasa adentro.

CONCEPTO · EL POTENCIÓMETRO

Es una resistencia con un cursor que se desliza. Con los extremos en 5V y GND, la pata del medio entrega un voltaje que depende de cuánto girás. Se llama **divisor de tensión**.

© José Gastón Cirvini

CUIDADO

- Conectar la señal a un pin digital: solo vas a leer 0 o 1.
- Olvidar `Serial.begin`: el Monitor queda vacío.
- Imprimir sin `delay`: los números pasan tan rápido que no se leen.

PROBÁ ADEMÁS

Mostrá el voltaje en lugar del número: `valor * 5.0 / 1023`.

EN LA PLACA REAL

Una lectura analógica real nunca está quieta: baila dos o tres unidades por el ruido eléctrico. Es normal.

Palabras nuevas: **analógico** **digital** **analogRead** **Monitor serie** **potenciómetro** **divisor de tensión**

5.2 Brillo regulable

META Dar a una salida una intensidad intermedia con PWM.

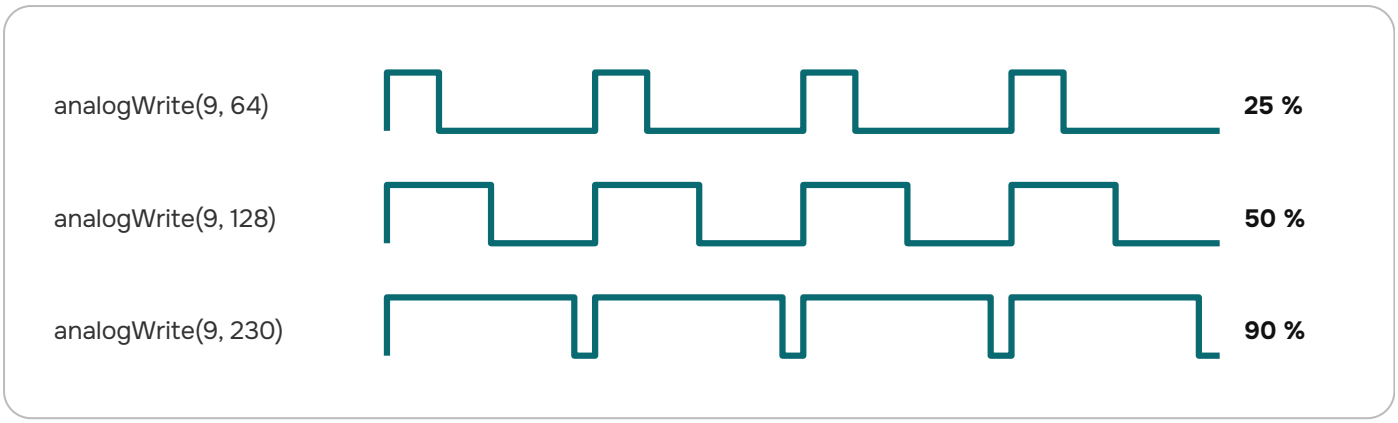
> Ya podés leer «cuánto». Falta poder responder «cuánto».

CONCEPTO

Un pin digital solo sabe entregar 0 V o 5 V. Para lograr un término medio usa un truco: se prende y se apaga unas 500 veces por segundo. Si pasa la mitad del tiempo encendido, el LED recibe la mitad de la energía y el ojo ve la mitad del brillo.

La técnica se llama **PWM** (modulación por ancho de pulso). `analogWrite(9, valor)` acepta de 0 (siempre apagado) a 255 (siempre encendido).

© José Gastón Cirvini



El voltaje siempre es 0 o 5 V. Lo que cambia es cuánto tiempo pasa en cada uno.

CUIDADO

- Usar un pin sin el símbolo `~`: solo los pines 3, 5, 6, 9, 10 y 11 hacen PWM.
- Pasarle un número mayor a 255: el valor «da la vuelta» y el brillo salta.
- Confundir los nombres: `analogWrite` no usa los pines A0–A5.

PROBÁ ADEMÁS

Sin potenciómetro: hacé que el LED se encienda de a poco usando varios `analogWrite` con valores crecientes.

EN LA PLACA REAL

PWM no entrega un voltaje intermedio de verdad. A un LED o a un motor les da igual; a un circuito que necesite 2,5 V reales, no.

Palabras nuevas: **PWM** **analogWrite** **ciclo de trabajo**

5.3 La función map

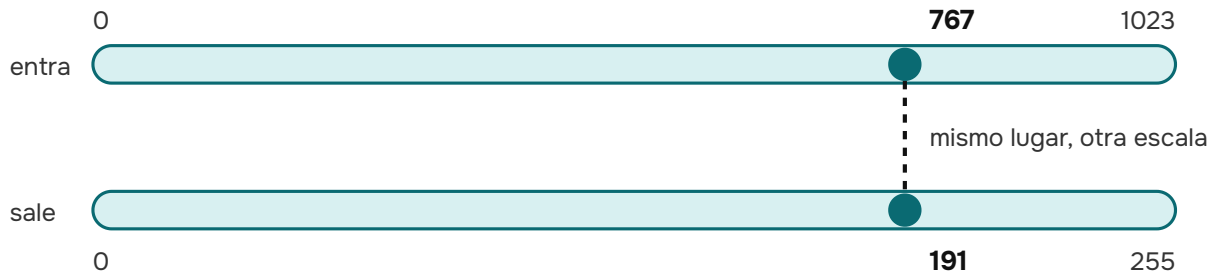
META Convertir un número de una escala a otra.

> En el nivel anterior, una línea del programa convertía la lectura en brillo. Esa línea usaba `map`. Ahora la escribís vos.

© José Gastón Cirvini

CONCEPTO

`map(valor, deMin, deMax, aMin, aMax)` es una regla de tres. Toma un valor que está en una escala y devuelve el que ocupa el mismo lugar en otra. Si el valor está a tres cuartos de la primera, el resultado está a tres cuartos de la segunda.



map no cambia la posición: cambia la regla con la que se mide.

QUIERO PASAR LA LECTURA A…	SE ESCRIBE
brillo de un LED	`map(v, 0, 1023, 0, 255)`
porcentaje	`map(v, 0, 1023, 0, 100)`
ángulo de un servo	`map(v, 0, 1023, 0, 180)`
escala invertida	`map(v, 0, 1023, 255, 0)`

CUIDADO

- Cambiar el orden de los cinco números: primero van los dos de la escala de entrada, después los dos de la de salida.
- `map` trabaja con enteros: descarta los decimales.
- No recorta: si el valor se sale de la escala de entrada, el resultado se sale de la de salida.

PROBÁ ADEMÁS

Invertí el control: perilla al máximo, LED apagado.

Palabras nuevas: **map** **escala**

© José Gastón Cirvini

5.4 Un umbral

META Convertir una medición en una decisión.

> Tenés un número entre 0 y 1023 y un `if`. Juntos hacen algo nuevo.

CONCEPTO

Un **umbral** es un valor límite que divide el rango en dos zonas. Por debajo pasa una cosa y por encima otra. Así una lectura con mil valores posibles termina en una decisión de sí o no.

Para comparar números hay seis **operadores**.

OPERADOR	PREGUNTA	EJEMPLO VERDADERO
`==`	¿es igual?	`5 == 5`
`!=`	¿es distinto?	`5 != 3`
`>`	¿es mayor?	`7 > 2`
`<`	¿es menor?	`2 < 7`
`>=`	¿es mayor o igual?	`5 >= 5`
`<=`	¿es menor o igual?	`4 <= 5`

EJEMPLO · AVISO DE VOLUMEN ALTO

```
int volumen = analogRead(A2);

if (volumen >= 800) {
  digitalWrite(13, HIGH);   // demasiado fuerte
} else {
  digitalWrite(13, LOW);
}
```

CUIDADO

- Elegir el umbral «a ojo»: primero mirá los valores reales en el Monitor serie.
- Justo en el umbral, el ruido hace que la salida titile. Se arregla con dos umbrales separados; lo vas a ver en el grupo 10.

© José Gastón Cirvini

PROBÁ ADEMÁS

Hacé que el LED se encienda solo en la zona del medio: entre 400 y 600.

Palabras nuevas: **umbral** **operador de comparación**

5.5 Barra de nivel

META Entender qué resuelven un arreglo y un `for`.

> Diez LED serían diez `pinMode` y veinte `digitalWrite`. Tiene que haber una forma mejor.

CONCEPTO

Cuando un programa repite la misma línea cambiando solo un número, hay dos herramientas que lo acortan. Acá las vas a ver funcionando; en el grupo 8 las vas a escribir vos.

Un **arreglo** es una fila de cajas con un solo nombre. Cada caja se identifica por su posición, que se cuenta **desde cero**.

Un `for` repite un bloque llevando la cuenta en una variable.

EJEMPLO · CUATRO LUCES DE UN TABLERO

```
int luces[] = {4, 5, 6, 7};      // luces[0] vale 4, luces[3] vale 7

for (int i = 0; i < 4; i++) {    // i vale 0, 1, 2 y 3
  pinMode(luces[i], OUTPUT);
}
```

CUIDADO

- Contar desde 1: la primera posición es la 0, y en un arreglo de 10 la última es la 9.
- Conectar los cables en otro orden: la barra enciende salteada.

PROBÁ ADEMÁS

En el modo libre, cambiá el orden de los números del arreglo y mirá qué pasa con la barra.

© José Gastón Cirvini

EN LA PLACA REAL

Una barra de LED real es solo diez LED en una misma cápsula. Cada uno necesita su resistencia; algunos módulos ya las traen.

Palabras nuevas: **arreglo** **índice** **for**

© José Gastón Cirvini

GRUPO 6 · ETAPA: SENSORES

Sensores del mundo real

Un sensor convierte algo del mundo (luz, calor, movimiento, distancia) en una señal que el Arduino puede leer. Con sensores, el programa reacciona a lo que pasa a su alrededor.

- Módulos de tres patas: VCC, GND y señal.
- Sensores analógicos y digitales.
- Tipos de datos nuevos: `float` y `long`.
- Qué es una librería y cómo se usa un objeto.

6.1 Luz automática

META Armar un automatismo con un sensor analógico y un umbral.

> Reemplazá la perilla del grupo 5 por algo que se mueva solo: la luz del ambiente.

CONCEPTO

Un **sensor** entrega una señal eléctrica que depende de una magnitud física. El de luz da más voltaje cuanta más luz recibe. Para el Arduino es igual que un potenciómetro: se lee con `analogRead`.

Casi todos los **módulos** traen tres patas: **VCC** (alimentación, 5 V), **GND** y **OUT** (la señal).

CONCEPTO · AUTOMATISMO

Sensor + umbral + actuador = un sistema que decide solo. Es la estructura de un farol de calle, una heladera o un termotanque: **medir, comparar, actuar**.

CUIDADO

- Olvidar VCC o GND del módulo: el sensor no mide nada.
- Dar por hecho el umbral: cada ambiente tiene otra luz. Medí primero.
- Poner el LED al lado del sensor: se ilumina a sí mismo y entra en un ciclo de prender y apagar.

© José Gastón Cirvini

PROBÁ ADEMÁS

Invertí el automatismo: que el LED se encienda cuando hay mucha luz.

EN LA PLACA REAL

El sensor de luz más común (LDR) es una pieza de dos patas, sin módulo. Se arma con una resistencia de 10 kΩ en serie, formando un divisor de tensión como el del potenciómetro.

Palabras nuevas: **sensor** **módulo** **VCC** **OUT** **automatismo**

6.2 Alarma de calor

META Convertir una lectura en una unidad real y usar números con coma.

> El sensor de luz daba un número sin unidad. La temperatura se mide en grados, y eso pide una cuenta.

CONCEPTO

El sensor entrega voltaje, no grados. El fabricante publica la relación en la **hoja de datos**. La conversión tiene dos pasos: de lectura a volts, y de volts a grados.

CONCEPTO · FLOAT

`int` guarda enteros y corta los decimales: 7 dividido 2 da 3. `float` guarda números con coma: 3,5. Para que una división conserve los decimales, al menos uno de los números tiene que llevar punto: `5.0` en lugar de `5`.

EJEMPLO · LA CUENTA PASO A PASO

Para un sensor que entrega 10 milivolts por grado, con medio volt en cero grados:

```
int lectura = analogRead(A1);          // por ejemplo, 153
float volts = lectura * 5.0 / 1023;    // 0.75 V
float grados = (volts - 0.5) * 100;    // 25 grados
```

CONCEPTO · UNA COMPARACIÓN YA ES UN VALOR

`grados > 30` vale verdadero o falso. Se puede usar directamente donde haga falta un HIGH o LOW: `digitalWrite(13, grados > 30);`.

CUIDADO

- Escribir `5 / 1023` en lugar de `5.0 / 1023`: da 0, y todo el resultado da 0.
- En código, los decimales llevan punto, no coma.

PROBÁ ADEMÁS

Mostrá en el Monitor serie la temperatura también en grados Fahrenheit: `grados * 1.8 + 32`.

EN LA PLACA REAL

Hay sensores parecidos con fórmulas distintas (TMP36, LM35). Antes de copiar una cuenta, fijate qué modelo tenés.

Palabras nuevas: **float** **hoja de datos** **conversión**

6.3 Detector de movimiento

META Usar un sensor digital y manejar dos salidas a la vez.

> No todos los sensores entregan un número. Algunos ya traen la decisión tomada.

CONCEPTO

Un **sensor digital** tiene un circuito propio que mide y compara. Su salida solo dice sí o no. El sensor de movimiento (PIR) detecta el calor de un cuerpo que se mueve y pone su salida en `HIGH`.

No necesita pull-down: el módulo siempre entrega 0 V o 5 V, nunca deja el pin flotando.

	SENSOR ANALÓGICO	SENSOR DIGITAL
Entrega	un voltaje variable	HIGH o LOW
Se lee con	`analogRead`	`digitalRead`
El umbral	lo ponés en el programa	ya viene en el sensor
Ejemplos	luz, temperatura, humedad de suelo	movimiento, toque, llama

CONCEPTO · TEXTO EN EL MONITOR SERIE

Un texto entre comillas se llama **cadena**. `Serial.println("Movimiento")` lo escribe tal cual. Sin comillas, el Arduino busca una variable con ese nombre.

CUIDADO

- Leer un sensor digital con `analogRead`: da 0 o 1023, y confunde.
- Olvidar las comillas del texto.

PROBÁ ADEMÁS

Hacé que la alarma siga sonando dos segundos después de que el movimiento termina.

EN LA PLACA REAL

Un PIR real tarda cerca de medio minuto en estabilizarse al encenderlo, y después de detectar mantiene la salida en alto unos segundos.

Palabras nuevas: **sensor digital** **PIR** **cadena de texto**

6.4 Sensor de estacionamiento

META Medir una distancia a partir de un tiempo.

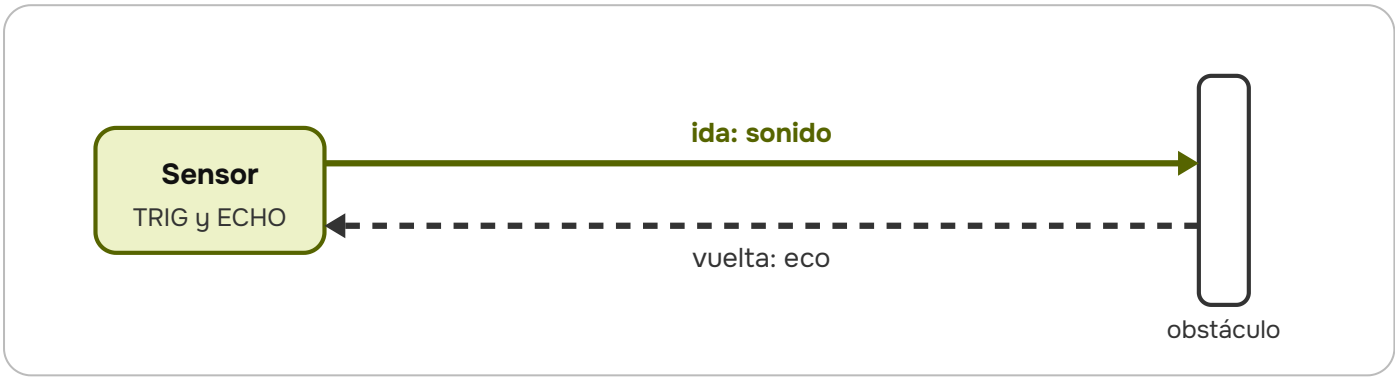
> Hasta acá los sensores entregaban el dato. Este hay que interrogarlo.

© José Gastón Cirvini

CONCEPTO

El sensor ultrasónico mide como un murciélago. Por **TRIG** el Arduino le ordena emitir un sonido demasiado agudo para el oído. Por **ECHO** el sensor avisa cuánto tardó el eco en volver.

`pulseIn(pin, HIGH)` mide cuánto dura ese aviso, en **microsegundos** (millonésimas de segundo).



El tiempo medido incluye la ida y la vuelta.

CONCEPTO · DE TIEMPO A DISTANCIA

El sonido recorre un centímetro en unos 29 microsegundos. Como el tiempo medido es de ida y vuelta, se divide por 2: en total, **distancia = tiempo ÷ 58**.

`long` es un tipo para enteros muy grandes. Un `int` llega hasta 32.767 y un eco lejano lo supera.

CUIDADO

- Cruzar TRIG y ECHO: la distancia da siempre 0.
- Guardar el tiempo en un `int`: con objetos lejanos el número se desborda y da valores absurdos.

PROBÁ ADEMÁS

Hacé que el zumbador suene más agudo cuanto más cerca está el obstáculo, usando `map`.

EN LA PLACA REAL

El sensor real mide bien entre 2 cm y unos 4 m, y falla con superficies blandas o inclinadas, que no devuelven el eco.

© José Gastón Cirvini

Palabras nuevas: **ultrasónico** **pulseIn** **microsegundo** **long**

6.5 Temperatura y humedad

META Usar una librería y pedirle datos a un objeto.

> Para el ultrasónico alcanzó una cuenta. Hay sensores que hablan en un idioma propio, imposible de leer a mano.

CONCEPTO

Una **librería** es código que escribió otra persona para resolver un problema concreto. Se suma al programa con `#include`. Gracias a ella no hace falta saber cómo se comunica el sensor: alcanza con pedirle el dato.

CONCEPTO · OBJETOS

`DHT dht(2, DHT11);` crea un **objeto**: una pieza de software que representa al sensor. Se llama `dht`, está en el pin 2 y es del modelo DHT11.

A un objeto se le piden cosas con un punto: `dht.readTemperature()`. Lo que va después del punto se llama **método**. Ya venías usando uno: `Serial` es un objeto y `println` es su método.

EJEMPLO · LEER Y ARMAR UN RENGLÓN

`print` escribe sin saltar de línea; `println` escribe y salta.

```
float h = dht.readHumidity();

Serial.print("Humedad: ");
Serial.print(h);
Serial.println(" %");       // Humedad: 54.00 %
```

CUIDADO

- Olvidar `dht.begin()` en `setup`: las lecturas no llegan.
- Leerlo muy seguido: necesita al menos un segundo entre lecturas.

© José Gastón Cirvini

PROBÁ ADEMÁS

Hacé que el LED se encienda si hace más de 28 grados *o* hay más de 70 % de humedad. Una pista: `||`.

EN LA PLACA REAL

En el Arduino IDE las librerías se instalan una vez, desde el Gestor de librerías, antes de poder usarlas.

Palabras nuevas: **librería** **#include** **objeto** **método**

© José Gastón Cirvini

GRUPO 7 · ETAPA: ACTUADORES

Cosas que se mueven y suenan

Un LED es solo el comienzo. Los actuadores de este grupo hacen ruido, giran, empujan y encienden aparatos. Cada uno pide algo distinto del Arduino.

- Sonido con frecuencia.
- Servos: movimiento con ángulo.
- Motores y relés: por qué necesitan un intermediario.
- Tu primera función propia.

7.1 Una melodía

META Producir sonidos de distinta altura con `tone`.

> En el grupo 2 el zumbador solo sabía sonar o callar. Con un programa, también puede cantar.

CONCEPTO

El sonido es una vibración. La cantidad de vibraciones por segundo se llama **frecuencia** y se mide en **hertz** (Hz). Pocas vibraciones dan un sonido grave; muchas, uno agudo.

`tone(pin, frecuencia)` hace vibrar el pin a ese ritmo y sigue sonando hasta que llega un `noTone(pin)` u otro `tone`.

NOTA	DO	RE	MI	FA	SOL	LA	SI	DO AGUDO
Hz	262	294	330	349	392	440	494	523

CONCEPTO · LA DURACIÓN LA PONE EL DELAY

`tone` arranca el sonido y el programa sigue de inmediato. Lo que hace durar la nota es el `delay` que viene después.

© José Gastón Cirvini

CUIDADO

- Olvidar `noTone`: la última nota suena para siempre.
- Dos `tone` seguidos sin `delay`: solo se escucha el segundo.

PROBÁ ADEMÁS

Escribí las primeras notas de una canción que conozcas. Cada vez que el número se duplica, la nota sube una octava.

EN LA PLACA REAL

Hay dos zumbadores parecidos: el activo suena solo con darle corriente y siempre igual; el pasivo necesita `tone` y puede dar notas.

Palabras nuevas: **frecuencia** **hertz** **tone** **noTone**

7.2 Servo a mano

META Mover un servo a un ángulo exacto.

> Hasta ahora las salidas daban luz o sonido. Esta se mueve.

CONCEPTO

Un **servo** es un motor con engranajes y un circuito de control. No gira libremente: va al ángulo que se le pide, entre 0 y 180 grados, y se queda ahí haciendo fuerza.

Tiene tres cables: alimentación, GND y señal. Por la señal recibe pulsos cuyo ancho indica el ángulo. De generarlos se ocupa la librería `Servo`.

© José Gastón Cirvini

EJEMPLO · UNA BARRERA QUE SUBE Y BAJA

```
#include <Servo.h>
Servo barrera;

void setup() {
  barrera.attach(5);     // la señal está en el pin 5
}

void loop() {
  barrera.write(90);     // arriba
  delay(2000);
  barrera.write(0);      // abajo
  delay(2000);
}
```

CUIDADO

- Olvidar `attach`: el servo no recibe señal.
- Pedir un ángulo mayor a 180 o menor a 0: el servo queda forzando contra el tope.
- Darle órdenes sin pausa: no llega a moverse antes de la siguiente.

PROBÁ ADEMÁS

Hacé que el servo recorra de 0 a 180 de a 10 grados, con una pausa en cada paso.

EN LA PLACA REAL

Un servo consume bastante corriente al moverse. Con uno chico alcanza el pin 5V; con varios o con uno grande hace falta una fuente aparte, compartiendo GND con el Arduino.

Palabras nuevas: **servo** **ángulo** **attach** **write**

7.3 Ventilador automático

META Controlar un motor con un módulo intermediario.

> El servo traía su propio circuito de control. Un motor común no.

CONCEPTO

Un pin del Arduino entrega como mucho unos 20 miliamperes. Un motor chico pide diez veces más. Conectarlo directo puede dañar la placa.

La solución es un **módulo controlador** (driver): un interruptor electrónico. El pin le da la orden con una corriente mínima y el módulo deja pasar hacia el motor la corriente grande, que viene de la alimentación.

Es la misma idea que la llave de luz de tu casa: tu dedo no enciende la lámpara, solo acciona algo que sí puede.

CONCEPTO · CONSTRAIN

`constrain(valor, minimo, maximo)` recorta un número para que quede dentro de los límites. Es el complemento de `map`, que no recorta.

EJEMPLO · VELOCIDAD SEGÚN UNA PERILLA

```
int perilla = analogRead(A0);
int velocidad = map(perilla, 0, 1023, 0, 255);
velocidad = constrain(velocidad, 0, 255);
analogWrite(5, velocidad);
```

CUIDADO

- Conectar un motor directo a un pin.
- Con valores PWM muy bajos el motor zumba y no arranca: no le alcanza la fuerza.

PROBÁ ADEMÁS

Hacé que el ventilador tenga tres velocidades fijas según la temperatura: apagado, medio y máximo.

EN LA PLACA REAL

Un motor genera golpes eléctricos al arrancar y al frenar. Los módulos traen un diodo de protección; un transistor suelto necesita que lo agregues.

Palabras nuevas: **motor** **controlador (driver)** **constrain**

© José Gastón Cirvini

7.4 Riego automático

META Encender aparatos grandes con un relé.

> El módulo del motor manejaba algo más grande que un LED. El relé maneja cosas mucho más grandes.

CONCEPTO

Un **relé** es un interruptor mecánico movido por un electroimán. Cuando el Arduino le da la orden, el imán atrae una lámina y cierra un circuito completamente separado. Por eso se escucha un «clic».

Los dos circuitos no se tocan: el Arduino trabaja con 5 V y el relé puede estar cerrando el paso de una bomba o de una lámpara.

CONCEPTO · PORCENTAJES

Un número como 387 no dice nada. Convertido con `map(lectura, 0, 1023, 0, 100)` pasa a ser 38 %, que se entiende a simple vista y permite escribir condiciones claras: `if (humedad < 30)`.

CUIDADO

- Prender y apagar un relé muchas veces por segundo: es mecánico y se desgasta.
- Usar PWM en un relé: no regula, solo lo destruye más rápido.

PROBÁ ADEMÁS

Agregá un LED que avise mientras la bomba está encendida.

EN LA PLACA REAL

Un relé puede manejar los 220 V de la red, y eso es peligroso. En el aula se usa solo con bajo voltaje. Cualquier trabajo con 220 V lo hace una persona adulta capacitada.

Palabras nuevas: **relé** **electroimán** **porcentaje**

© José Gastón Cirvini

7.5 Mezclar colores

META Escribir y usar una función propia con parámetros.

> Para mostrar un color hay que escribir tres `analogWrite`. Para mostrar cuatro colores, doce. Es momento de ponerle nombre a ese grupo de líneas.

CONCEPTO

Un LED RGB tiene adentro un LED rojo, uno verde y uno azul. Con tres valores PWM se obtiene cualquier mezcla: rojo y verde dan amarillo; los tres al máximo dan blanco.

CONCEPTO · FUNCIONES

Una **función** es un bloque de código con nombre. Se escribe una vez y se usa cuantas veces haga falta. Ya usabas funciones hechas por otros (`delay`, `digitalWrite`); ahora hacés la tuya.

Los **parámetros** son los datos que la función necesita para trabajar. Van entre paréntesis, cada uno con su tipo.

EJEMPLO · UNA FUNCIÓN PARA DESTELLAR

```
void destello(int pin, int tiempo) {   // así se define
  digitalWrite(pin, HIGH);
  delay(tiempo);
  digitalWrite(pin, LOW);
}

void loop() {
  destello(13, 100);                   // así se usa
  destello(12, 400);
  delay(1000);
}
```

CUIDADO

- Definir una función dentro de otra: van una al lado de la otra, nunca anidadas.
- Pasar los valores en otro orden que el de los parámetros.
- Compartir una sola resistencia entre los tres colores: se roban corriente y el color cambia.

© José Gastón Cirvini

PROBÁ ADEMÁS

Agregá el color naranja. ¿Qué tres números necesita?

EN LA PLACA REAL

Hay LED RGB de cátodo común (la pata larga va a GND) y de ánodo común (va a 5V y la lógica se invierte). Mirá cuál tenés antes de cablear.

Palabras nuevas: **RGB** **función** **parámetro** **void**

GRUPO 8 · ETAPA: PROGRAMACIÓN INTERMEDIA

Programar en serio

Ya sabés hacer que las cosas funcionen. Este grupo trata de escribir programas que además se puedan leer, corregir y agrandar. Los circuitos vienen armados: toda la atención va al código.

- Constantes con nombre.
- El `for` y los arreglos, escritos por vos.
- Funciones con parámetros.
- Medir el tiempo sin detener el programa.

8.1 Constantes con nombre

META **Reemplazar números sueltos por nombres.**

> En el nivel 3.3 cambiaste un pin a mano en tres lugares. Era fácil olvidarse de uno.

CONCEPTO

Un número que aparece en el código sin explicación se llama **número mágico**. Tiene dos problemas: no se sabe qué significa, y si hay que cambiarlo hay que encontrarlo en todos los lugares.

Una **constante** le pone nombre a un valor que no cambia: `const int LED = 13;`. A partir de ahí se escribe `LED`. Para mudar el LED se cambia una sola línea.

	VARIABLE	CONSTANTE
Se declara	`int toques = 0;`	`const int LED = 13;`
¿Puede cambiar?	sí, mientras el programa corre	no
Se usa para	datos que varían: lecturas, contadores	datos fijos: pines, umbrales, tiempos
Costumbre	minúsculas	MAYÚSCULAS

© José Gastón Cirvini

EJEMPLO · ANTES Y DESPUÉS

```
// antes
if (analogRead(A0) < 400) { digitalWrite(13, HIGH); }

// después
const int SENSOR_LUZ = A0;
const int FAROL = 13;
const int OSCURO = 400;

if (analogRead(SENSOR_LUZ) < OSCURO) { digitalWrite(FAROL, HIGH); }
```

CUIDADO

- Intentar cambiar una constante más abajo: el programa no se carga.
- Nombres como `x` o `cosa`: no dicen nada.
- Espacios o tildes en los nombres: no están permitidos.

PROBÁ ADEMÁS

Convertí también los tiempos de espera en constantes.

Palabras nuevas: **constante** **const** **número mágico**

8.2 Repetir con for

META Escribir un `for` y saber cuántas vueltas da.

> En el nivel 5.5 viste un `for` funcionando. Ahora lo escribís.

CONCEPTO

Un `for` tiene tres partes entre paréntesis, separadas por punto y coma.

© José Gastón Cirvini

PARTE	EJEMPLO	QUÉ HACE
Inicio	`int i = 0`	crea el contador y le da su primer valor
Condición	`i < 5`	se revisa antes de cada vuelta; si es falsa, termina
Avance	`i++`	se ejecuta al final de cada vuelta; suma uno

EJEMPLO · SEGUIR UN FOR A MANO

Este `for` da cinco vueltas. Hacer la tabla es la forma de estar seguro.

```
for (int i = 0; i < 5; i++) {
  Serial.println(i * 10);     // escribe 0, 10, 20, 30, 40
}
```

CONCEPTO · MENOR O MENOR-IGUAL

`i < 5` empezando en 0 da cinco vueltas (0 a 4). `i ⩽ 5` da seis (0 a 5). Equivocarse por uno es el error más común de la programación, tanto que tiene nombre: *off-by-one*.

CUIDADO

- Separar las partes con comas en lugar de punto y coma.
- Una condición que nunca se vuelve falsa: el `for` no termina más.
- Usar el contador fuera del `for`: solo existe adentro.

PROBÁ ADEMÁS

Hacé que los LED se enciendan de dos en dos: `pin = pin + 2`.

Palabras nuevas: **contador** **i++** **bucle**

8.3 Arreglos: ida y vuelta

META Recorrer un arreglo en los dos sentidos.

© José Gastón Cirvini

> En el nivel anterior los pines eran consecutivos y el contador servía de número de pin. Casi nunca hay tanta suerte.

CONCEPTO

Un arreglo guarda los valores en el orden que quieras. El `for` ya no recorre pines sino **posiciones**, y el arreglo dice qué pin hay en cada una.

En un arreglo de 4 elementos las posiciones válidas son 0, 1, 2 y 3. La posición 4 no existe.

EJEMPLO · PINES SALTEADOS Y CUENTA REGRESIVA

`i--` resta uno. Para ir hacia atrás, se arranca en la última posición y se sigue mientras `i` sea mayor o igual a cero.

```
int filas[] = {3, 7, 12};

for (int i = 2; i ⩾ 0; i--) {     // 2, 1, 0
  digitalWrite(filas[i], HIGH);
  delay(300);
}
```

CUIDADO

- Leer una posición que no existe: el Arduino no avisa, devuelve basura. Es de los errores más difíciles de encontrar.
- Empezar la vuelta en la posición 4 de un arreglo de 4.

PROBÁ ADEMÁS

Agregá un quinto LED al arreglo. ¿Qué números del programa hay que cambiar? ¿Cómo harías para cambiar uno solo?

Palabras nuevas: **posición** **i--** **recorrer**

8.4 Funciones propias

META Dividir un programa en funciones con parámetros.

© José Gastón Cirvini

> En el nivel 7.5 usaste una función que ya estaba escrita. Ahora completás una y decidís cómo usarla.

CONCEPTO

Una función bien hecha resuelve **una sola cosa** y su nombre dice cuál. El `loop` que la usa se lee como una lista de tareas.

Dentro de la función, los parámetros se usan como variables. Su valor lo pone quien la llama: en `parpadear(13, 3)`, `pin` vale 13 y `veces` vale 3. En la llamada siguiente pueden valer otra cosa.

EJEMPLO · EL MISMO PROGRAMA, ANTES Y DESPUÉS

```
void loop() {            // sin funciones: hay que leer todo para entender
  tone(8, 880); delay(100); noTone(8); delay(100);
  tone(8, 880); delay(100); noTone(8); delay(100);
}

void loop() {            // con funciones: se entiende de un vistazo
  pitido();
  pitido();
}
```

CUIDADO

- Escribir el número de pin dentro de la función en lugar de usar el parámetro: deja de servir para otros pines.
- Llamar a la función sin paréntesis: `pitido;` no hace nada.

PROBÁ ADEMÁS

Agregá un tercer parámetro para elegir la velocidad del parpadeo.

Palabras nuevas: **llamar a una función** **reutilizar**

8.5 Dos cosas a la vez

META Medir el tiempo con `millis` sin frenar el programa.

© José Gastón Cirvini

> `delay` te acompañó desde el grupo 3. Tiene un defecto que hasta ahora no importaba.

CONCEPTO

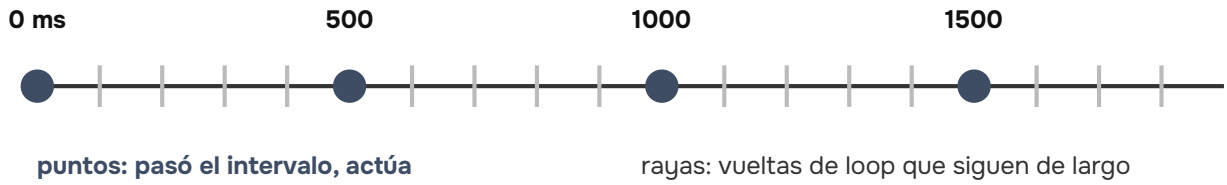
Durante un `delay(1000)` el Arduino queda sordo y ciego un segundo entero: no lee botones ni sensores. En un programa que hace una sola cosa no molesta. En uno que hace dos, lo arruina.

`millis()` devuelve cuántos milisegundos pasaron desde que el Arduino arrancó. Es un reloj que nunca se detiene.

CONCEPTO · MIRAR EL RELOJ

En lugar de esperar, el programa anota a qué hora hizo algo por última vez. En cada vuelta pregunta: ¿ya pasó el intervalo? Si no, sigue de largo y atiende lo demás. Si sí, actúa y anota la hora nueva.

Es lo que hacés al cocinar: no te quedás mirando la olla, mirás el reloj cada tanto mientras hacés otra cosa.



puntos: pasó el intervalo, actúa — rayas: vueltas de loop que siguen de largo

El loop da miles de vueltas. Solo en algunas se cumple el intervalo.

EJEMPLO · IMPRIMIR CADA DOS SEGUNDOS SIN FRENAR

```
unsigned long ultima = 0;

void loop() {
  if (millis() - ultima ⩾ 2000) {
    ultima = millis();
    Serial.println("pasaron 2 segundos");
  }
  // acá el programa sigue atendiendo todo lo demás
}
```

© José Gastón Cirvini

CUIDADO

- Guardar `millis()` en un `int`: se desborda a los 32 segundos. Va en `unsigned long`.
- Olvidar actualizar la hora guardada: la condición queda verdadera para siempre.
- Dejar un `delay` largo en otra parte del `loop`: vuelve a frenar todo.

PROBÁ ADEMÁS

Hacé parpadear dos LED con ritmos distintos (300 ms y 700 ms), cada uno con su propia variable de tiempo.

Palabras nuevas: **millis** **unsigned long** **intervalo** **desborde**

© José Gastón Cirvini

GRUPO 9 · ETAPA: PANTALLAS

Pantallas

Un proyecto que no muestra nada es difícil de usar y de corregir. De un renglón de texto a un gráfico: cinco formas de que el Arduino se comunique con las personas.

- El Monitor serie como herramienta de trabajo.
- Cómo se forma un número con segmentos.
- Módulos que se manejan con dos cables.
- El bus I2C y las coordenadas de una pantalla.

9.1 Hablar por el Monitor serie

META Usar el Monitor serie para ver qué hace el programa.

> Venís usando `Serial` desde el grupo 5 para leer sensores. Sirve para mucho más.

CONCEPTO

Buscar y corregir errores se llama **depurar**. La técnica más simple es hacer que el programa cuente lo que está haciendo: imprimir el valor de una variable justo antes de una decisión muestra por qué tomó ese camino.

INSTRUCCIÓN	QUÉ ESCRIBE	DESPUÉS
`Serial.print("Luz: ")`	el texto tal cual	se queda en el renglón
`Serial.print(valor)`	el contenido de la variable	se queda en el renglón
`Serial.println(valor)`	el contenido de la variable	salta al renglón siguiente
`Serial.println()`	nada	solo salta de renglón

CONCEPTO · SUMAR UNO

`vuelta++` es la forma corta de `vuelta = vuelta + 1`. Una variable que crece de a uno es un **contador**.

© José Gastón Cirvini

CUIDADO

- Poner comillas al nombre de una variable: se imprime el nombre, no el valor.
- Declarar el contador dentro de `loop`: vuelve a cero en cada vuelta.

PROBÁ ADEMÁS

Mostrá también cuántos segundos lleva encendido el Arduino: `millis() / 1000`.

EN LA PLACA REAL

En el Arduino IDE la velocidad elegida en el Monitor tiene que coincidir con la de `Serial.begin`. Si no, aparecen símbolos sin sentido.

Palabras nuevas: **depurar** **contador** **print / println**

9.2 Contador de 7 segmentos

META Entender cómo se guarda un dibujo en un número.

> El Monitor serie necesita una computadora. Este display muestra números por sí solo.

CONCEPTO

Un display de 7 segmentos son siete LED alargados, nombrados de la **a** a la **g**. Cada dígito es una combinación: el 1 usa b y c; el 8 usa los siete.

CONCEPTO · BITS Y BYTES

Un **bit** es el dato más chico que existe: 0 o 1, apagado o encendido. Un **byte** son ocho bits juntos.

Como cada segmento también es apagado o encendido, un byte alcanza para guardar un dígito completo: un bit por segmento. El programa guarda diez bytes en un arreglo, uno por número, y `bitRead(dato, i)` lee el bit de la posición `i`.

© José Gastón Cirvini

DÍGITO	SEGMENTOS ENCENDIDOS
1	b c
4	b c f g
7	a b c
8	a b c d e f g

CUIDADO

- Cruzar dos cables: los números salen deformados, siempre de la misma manera.
- Olvidar el cable a GND: no enciende nada.

PROBÁ ADEMÁS

Modificá el programa para que cuente hacia atrás, de 9 a 0.

EN LA PLACA REAL

Un display suelto necesita una resistencia por segmento. Y hay dos tipos, cátodo común y ánodo común, que se conectan al revés.

Palabras nuevas: **segmento** **bit** **byte** **bitRead**

9.3 Reloj de 4 dígitos

META Manejar un módulo con dos cables y separar un número en partes.

> Un dígito pidió siete cables. Cuatro dígitos pedirían veintiocho.

CONCEPTO

Para no ocupar todos los pines, muchos módulos traen un chip que hace el trabajo pesado. El Arduino solo le envía datos por dos cables: **CLK** (reloj) marca el ritmo y **DIO** (datos) lleva la información, un bit por vez. Enviar los datos en fila por un solo cable se llama **comunicación serie**.

© José Gastón Cirvini

CONCEPTO · DIVISIÓN ENTERA Y RESTO

Con enteros, `/` da el cociente sin decimales y `%` da el **resto**. Juntos sirven para partir un número.

EJEMPLO · 135 SEGUNDOS SON 2 MINUTOS Y 15 SEGUNDOS

```
int total = 135;
int minutos = total / 60;      // 2
int segundos = total % 60;     // 15
```

CUIDADO

- Cruzar CLK y DIO: la pantalla queda en blanco.
- Confundir `%` con porcentaje: en código es el resto de la división.

PROBÁ ADEMÁS

Usá `%` para saber si un número es par: el resto de dividirlo por 2 es 0.

Palabras nuevas: **CLK** **DIO** **comunicación serie** **resto (%)**

9.4 Tu nombre en la pantalla LCD

META Escribir texto en una posición exacta de la pantalla.

> Los displays anteriores solo mostraban números. Esta pantalla muestra letras.

CONCEPTO

La pantalla LCD de este nivel tiene 2 filas de 16 caracteres. Cada lugar se identifica con dos números: columna (0 a 15) y fila (0 o 1). `lcd.setCursor(columna, fila)` elige dónde empezar y `lcd.print` escribe desde ahí.

CONCEPTO · EL BUS I2C

I2C es un sistema de dos cables, **SDA** (datos) y **SCL** (reloj), al que se pueden conectar muchos dispositivos a la vez. Cada uno tiene una **dirección**, como el número de una casa. En `lcd(0x27, 16, 2)`, `0x27` es la dirección de la pantalla. En el Arduino UNO, SDA está en A4 y SCL en A5.

EJEMPLO · UN DATO QUE CAMBIA

Lo que se escribe queda en pantalla hasta que se pisa. Los espacios del final borran restos de un número más largo.

```
lcd.setCursor(0, 1);
lcd.print("Luz: ");
lcd.print(analogRead(A0));
lcd.print("    ");
```

CUIDADO

- Invertir el orden: primero va la columna, después la fila.
- Escribir más de 16 caracteres: el resto no se ve.
- Olvidar `lcd.init()` o `lcd.backlight()`.

PROBÁ ADEMÁS

Centrá tu nombre en la fila de abajo. ¿En qué columna tiene que empezar?

EN LA PLACA REAL

La dirección puede ser 0x27 o 0x3F según el fabricante. Y el módulo trae una perilla azul para el contraste: si la pantalla parece vacía, girala.

Palabras nuevas: **LCD** **I2C** **SDA / SCL** **dirección** **cursor**

9.5 Gráficos en la pantalla OLED

META Dibujar con coordenadas.

© José Gastón Cirvini

> La LCD tenía 32 casilleros fijos. Esta pantalla tiene más de ocho mil puntos que se encienden por separado.

CONCEPTO

Cada punto se llama **píxel**. La pantalla tiene 128 de ancho y 64 de alto. Un píxel se ubica con dos **coordenadas**: `x` crece hacia la derecha e `y` crece hacia **abajo**. El punto (0, 0) es la esquina de arriba a la izquierda.

CONCEPTO · DIBUJAR EN BORRADOR

Las instrucciones de dibujo trabajan sobre una copia en memoria. Recién `display.display()` la envía a la pantalla. El ciclo de cada vuelta es siempre el mismo: **borrar, dibujar, mostrar**.

INSTRUCCIÓN	DIBUJA
`drawPixel(x, y, WHITE)`	un punto
`drawLine(x1, y1, x2, y2, WHITE)`	una línea
`drawRect(x, y, ancho, alto, WHITE)`	un rectángulo vacío
`fillRect(x, y, ancho, alto, WHITE)`	un rectángulo lleno
`drawCircle(x, y, radio, WHITE)`	un círculo

CUIDADO

- Olvidar `display.display()`: nada aparece.
- Olvidar `clearDisplay()`: los dibujos se acumulan.
- Pensar que `y` crece hacia arriba, como en matemática.

PROBÁ ADEMÁS

Dibujá un marco alrededor de toda la pantalla y un círculo en el centro.

EN LA PLACA REAL

Muchas pantallas OLED trabajan con 3,3 V. Mirá qué dice el módulo antes de conectarlo a 5V.

© José Gastón Cirvini

Palabras nuevas: **píxel** **coordenadas** **clearDisplay** **display**

© José Gastón Cirvini

GRUPO 10 · ETAPA: PROYECTOS

Proyectos completos

No queda casi nada nuevo por aprender: queda aprender a juntarlo. Un proyecto grande no se escribe de corrido; se divide en partes chicas que ya sabés resolver.

- Combinar condiciones.
- Leer un programa completo.
- Varias zonas con `else if`.
- Funciones que devuelven un resultado.
- Un método para encarar tus propios proyectos.

10.1 Alarma con llave

META Combinar varias condiciones en una sola decisión.

> En el grupo 2 armaste una Y con dos interruptores en serie. Es hora de escribirla.

CONCEPTO

Los **operadores lógicos** unen condiciones.

OPERADOR	SE LEE	ES VERDADERO CUANDO	EN CABLES
`&&`	Y	las dos condiciones se cumplen	serie
`||`	O	al menos una se cumple	paralelo
`!`	NO	la condición es falsa	—

EJEMPLO · COMBINAR LAS TRES

Se pueden mezclar, usando paréntesis para dejar claro el orden.

```
if ((luz < 300 || movimiento == HIGH) && !pausa) {
  digitalWrite(FAROL, HIGH);
}
```

© José Gastón Cirvini

CONCEPTO · LEERLO EN VOZ ALTA

Una condición larga se entiende mejor dicha en palabras: «si está oscuro o hay movimiento, y además no está en pausa». Si no se puede decir, conviene partirla en variables con nombre.

CUIDADO

- Escribir un solo `&` o un solo `|`: existen, pero hacen otra cosa.
- Olvidar que con `INPUT_PULLUP` la llave cerrada es `LOW`.

PROBÁ ADEMÁS

Agregá un pulsador de pánico: la alarma suena si hay movimiento con la llave puesta, o si alguien aprieta el pulsador.

Palabras nuevas: **operador lógico** **&&** **||**

10.2 Estación del clima

META Reconocer las partes de un programa completo antes de armarlo.

> Hasta ahora leías código de a pedazos. Este programa hay que leerlo entero.

CONCEPTO

Todo sistema se puede describir con tres bloques: **entradas** (lo que mide), **proceso** (lo que decide) y **salidas** (lo que hace o muestra). Antes de tocar un cable, hacé la lista.

BLOQUE	EN LA ESTACIÓN DEL CLIMA
Entradas	sensor de temperatura y humedad
Proceso	leer cada medio segundo y armar los textos
Salidas	pantalla LCD

© José Gastón Cirvini

CONCEPTO · LEER UN PROGRAMA AJENO

Un orden que funciona: primero las librerías y los objetos (qué componentes hay y en qué pines), después `setup` (qué se prepara) y por último `loop` (qué se repite). Los pines que aparecen son tu lista de conexiones.

CUIDADO

- Cablear todo junto y probar al final: si falla, no sabés dónde.
- Conectar primero la pantalla y comprobarla es más seguro: después te sirve para ver el resto.

PROBÁ ADEMÁS

Agregá un LED que se encienda si la humedad pasa de 80 %. ¿En cuál de los tres bloques entra?

EN LA PLACA REAL

Dos módulos I2C comparten los mismos pines A4 y A5 sin problema, mientras tengan direcciones distintas.

Palabras nuevas: **entrada / proceso / salida** **diagrama de bloques**

10.3 Maceta inteligente

META Manejar tres o más situaciones con `else if`.

> Con un umbral había dos zonas. Una planta puede estar seca, bien o ahogada: son tres.

CONCEPTO

`else if` agrega una pregunta más a la cadena. El programa revisa las condiciones de arriba hacia abajo y ejecuta **solo el primer bloque** cuya condición sea verdadera. El `else` final atrapa todo lo que no entró antes.

Como se detiene en la primera que se cumple, el **orden importa**.

© José Gastón Cirvini

EJEMPLO · UN TERMÓMETRO CON TRES LUCES

```
if (grados < 18) {
  mostrar(AZUL);            // frío
} else if (grados < 26) {
  mostrar(VERDE);           // agradable: entre 18 y 26
} else {
  mostrar(ROJO);            // calor
}
```

CONCEPTO · DOS UMBRALES EVITAN EL TITILEO

En el nivel 5.4 quedó un problema: justo en el umbral, la salida titila. Si la bomba se enciende por debajo de 30 % y no se apaga hasta superar un valor más alto, deja de dudar. Esa separación se llama **histéresis**.

CUIDADO

- Escribir tres `if` sueltos en lugar de una cadena: pueden cumplirse dos a la vez.
- Poner primero la condición más amplia: las siguientes nunca se alcanzan.

PROBÁ ADEMÁS

Agregá una cuarta zona: «muy seca», por debajo de 10 %, en la que además suene el zumbador.

EN LA PLACA REAL

Los sensores de humedad de suelo con puntas de metal se oxidan en pocas semanas. Los capacitivos duran mucho más.

Palabras nuevas: **else if** **histéresis** **zona**

10.4 Estacionamiento con luces

META Escribir una función que devuelve un resultado.

> Las funciones que hiciste hasta acá hacían algo. Esta además responde algo.

© José Gastón Cirvini

CONCEPTO

La palabra que va antes del nombre de una función indica qué devuelve. `void` significa «nada». `int` significa que entrega un número entero.

`return` termina la función y entrega el resultado al lugar desde donde fue llamada. Ese resultado se puede guardar en una variable o usar directo en una condición.

EJEMPLO · UNA FUNCIÓN QUE CONVIERTE

```
int porcentaje(int lectura) {
  return map(lectura, 0, 1023, 0, 100);
}

void loop() {
  int luz = porcentaje(analogRead(A0));
  if (porcentaje(analogRead(A1)) > 80) { /* ... */ }
}
```

CONCEPTO · CAJAS NEGRAS

Una vez que `medir()` funciona, no hace falta recordar cómo. Alcanza con saber qué entrega. Esa idea, usar algo por lo que hace sin pensar en cómo está hecho, es lo que permite construir cosas grandes. Es lo mismo que hacés con `delay` o con una librería.

CUIDADO

- Olvidar el `return` en una función que no es `void`.
- Declarar `void` y querer usar el resultado.
- Escribir código después del `return`: no se ejecuta nunca.

PROBÁ ADEMÁS

Escribí una función `bool estaCerca(int cm)` que devuelva verdadero si la distancia es menor a 30.

Palabras nuevas: **return** **valor de retorno** **caja negra**

© José Gastón Cirvini

10.5 Proyecto final: casa inteligente

META Encarar un proyecto dividiéndolo en partes.

> Todo lo que aprendiste entra en este nivel. Lo nuevo no es una instrucción: es una forma de trabajar.

CONCEPTO

La casa parece complicada: seis componentes y un programa largo. Mirándola de cerca son tres proyectos que ya hiciste: una luz automática (6.1), un ventilador (7.3) y una alarma (6.3). Cada uno vive en su función y no sabe nada de los otros.

CONCEPTO · UN MÉTODO PARA TUS PROYECTOS

1. **Describí** en una frase qué tiene que hacer.
2. **Listá** entradas, proceso y salidas.
3. **Dividí** en partes que se puedan probar por separado.
4. **Armá y probá una parte por vez**, con el Monitor serie a mano.
5. **Juntá** de a una. Si algo se rompe, el problema está en lo último que sumaste.
6. **Ordená**: constantes con nombre, una función por tarea, comentarios donde haga falta.

SISTEMA	ENTRADA	DECISIÓN	SALIDA
Luz	sensor de luz	menos de 400	LED
Ventilador	DHT11	más de 28 grados	motor
Alarma	sensor de movimiento	detecta a alguien	zumbador

CUIDADO

- Usar `delay` largos en una función: frena a las otras dos. Si hace falta esperar, se usa `millis`.
- Reutilizar un pin sin querer: una tabla de pines evita el problema.

PROBÁ ADEMÁS

Diseñá un cuarto sistema para la casa. Escribí primero la fila de la tabla y después la función.

© José Gastón Cirvini

EN LA PLACA REAL

Seis componentes juntos pueden pedir más corriente de la que entrega el cable USB. Los proyectos grandes usan una fuente aparte para motores y servos.

Palabras nuevas: **dividir un problema** **integrar** **probar por partes**

Glosario

Abierto / cerrado
Un circuito abierto está cortado y no conduce; uno cerrado está completo y conduce.

Actuador
Componente que hace algo en el mundo: luz, sonido, movimiento.

Algoritmo
Lista ordenada de pasos para resolver un problema, escrita en palabras.

Alimentar
Darle energía a un componente conectándolo a 5V y GND.

Analógico
Que puede tomar cualquier valor dentro de un rango, no solo dos.

analogRead
Lee el voltaje de un pin analógico y lo convierte en un número de 0 a 1023.

Ánodo
Pata positiva de un LED. Es la más larga.

Argumento
Dato que se le pasa a una instrucción entre paréntesis.

Arreglo
Fila de valores guardados bajo un mismo nombre. Las posiciones se cuentan desde cero.

Automatismo
Sistema que mide, compara y actúa sin intervención de una persona.

Bit
El dato más chico: 0 o 1.

Bloque
Conjunto de instrucciones entre llaves.

bool
Tipo de dato que solo guarda verdadero o falso.

Byte
Ocho bits juntos.

Cadena
Texto entre comillas.

Cargar
Enviar el programa a la placa para que empiece a ejecutarlo.

Cátodo
Pata negativa de un LED. Va hacia GND.

Circuito cerrado
Camino completo por el que la corriente sale y vuelve.

Comentario
Nota dentro del código, después de `//`, que el Arduino ignora.

Compilar
Traducir el código al idioma de la placa. Ahí aparecen los errores de escritura.

© José Gastón Cirvini

Condición
Pregunta que solo puede ser verdadera o falsa.

Contador
Variable que lleva la cuenta de algo.

Coordenadas
Par de números (x, y) que ubican un punto en una pantalla.

Cortocircuito
Unión directa entre positivo y GND, sin nada que frene la corriente.

Digital
Que solo puede tomar dos valores.

Diodo
Componente que deja pasar la corriente en un solo sentido.

Entrada
Componente que le avisa al sistema qué pasa afuera: un botón, un sensor.

Flanco
El instante en que una señal cambia de valor.

for
Estructura que repite un bloque llevando la cuenta.

Función
Bloque de código con nombre, que se escribe una vez y se usa muchas.

Hardware
La parte física de un proyecto: placa, cables y componentes.

Histéresis
Separación entre el umbral de encendido y el de apagado, para evitar titileos.

Constante
Valor con nombre que no cambia mientras el programa corre.

Controlador (driver)
Módulo que maneja corrientes grandes obedeciendo una señal chica.

Corriente
Cantidad de carga que pasa por segundo. Se mide en amperes (A).

Depurar
Buscar y corregir los errores de un programa.

digitalRead
Lee un pin digital: devuelve HIGH o LOW.

else if
Agrega una condición más a una cadena de decisiones.

Estado
Situación en la que está algo en un momento: abierto o cerrado, encendido o apagado.

float
Tipo de dato para números con decimales.

Frecuencia
Cantidad de vibraciones por segundo. Se mide en hertz (Hz).

GND
Tierra: el punto por donde la corriente vuelve. Vale 0 V.

HIGH / LOW
Los dos valores de un pin digital: 5 V y 0 V.

I2C
Sistema de comunicación de dos cables (SDA y SCL) para varios dispositivos.

© José Gastón Cirvini

if
Ejecuta un bloque solo si una condición es verdadera.

int
Tipo de dato para números enteros.

Ley de Ohm
Corriente = voltaje ÷ resistencia.

long
Tipo de dato para enteros muy grandes.

map
Convierte un número de una escala a otra.

Microcontrolador
Computadora diminuta dentro del Arduino que ejecuta el programa.

Milisegundo
Milésima parte de un segundo. Unidad que usa `delay`.

Módulo
Sensor o actuador montado en una plaquita lista para conectar.

O lógico (OR)
Verdadero si al menos una condición se cumple. En cables: paralelo.

Operador lógico
Une condiciones: && (Y), || (O), ! (NO).

Parámetro
Dato que una función recibe para trabajar.

Pin flotante
Entrada que no está conectada a nada y lee valores al azar.

Índice
Número de posición dentro de un arreglo.

Interruptor
Llave que se queda en la posición en que se la deja.

Librería
Código ya escrito que se suma al programa con #include.

loop
Bloque del programa que se repite para siempre.

Método
Acción que se le pide a un objeto con un punto.

Miliampere (mA)
Milésima parte de un ampere. Un LED usa entre 5 y 20 mA.

millis
Devuelve los milisegundos transcurridos desde que arrancó el Arduino.

Monitor serie
Ventana donde se ve el texto que el Arduino envía a la computadora.

Objeto
Pieza de software que representa a un componente y ofrece métodos.

Paralelo
Conexión en la que cada componente tiene su propio camino.

Pin digital
Contacto del Arduino que el programa pone en 5 V o 0 V.

Píxel
Cada punto de una pantalla.

© José Gastón Cirvini

Polarizado
Componente que solo funciona conectado en un sentido.

Protoboard
Placa con agujeros para armar circuitos sin soldar.

Pull-down
Resistencia que mantiene una entrada en 0 V mientras no se la usa.

Pulsador
Botón que cierra el circuito solo mientras se lo aprieta.

Rebote
Contactos falsos que hace un pulsador real durante unos milisegundos.

Resistencia
Componente que limita la corriente. Se mide en ohm (Ω).

return
Termina una función y entrega su resultado.

Secuencia
Instrucciones que se ejecutan una después de la otra, en orden.

Serie
Conexión en fila: la corriente pasa por un componente y después por el otro.

setup
Bloque del programa que se ejecuta una sola vez al encender.

Tabla de verdad
Tabla con todos los casos posibles de una regla lógica.

unsigned long
Tipo de dato para enteros positivos muy grandes, como los de millis.

Potenciómetro
Resistencia variable con una perilla.

Prueba de escritorio
Seguir un programa línea por línea anotando el estado de cada salida.

Pull-up
Resistencia que mantiene una entrada en 5 V mientras no se la usa.

PWM
Técnica que simula una intensidad intermedia prendiendo y apagando muy rápido.

Relé
Interruptor mecánico accionado por un electroimán.

Resto (%)
Lo que sobra de una división entera.

Riel
Fila larga de la protoboard, unida de punta a punta, para repartir energía.

Sensor
Componente que convierte una magnitud física en una señal eléctrica.

Servo
Motor que se mueve hasta un ángulo indicado y lo mantiene.

Software
El programa: las instrucciones que ejecuta la placa.

Umbral
Valor límite que separa dos comportamientos.

Variable
Caja con nombre donde el programa guarda un dato que puede cambiar.

© José Gastón Cirvini

Variable global
Variable declarada fuera de las funciones, que conserva su valor entre vueltas.

void
Indica que una función no devuelve ningún resultado.

Y lógico (AND)
Verdadero solo si todas las condiciones se cumplen. En cables: serie.

VCC
Pata de alimentación de un módulo.

Voltaje
Fuerza que empuja la corriente. Se mide en volts (V).

REALIZADO POR

José Gastón Cirvini

Mendoza, Argentina

DONACIONES

¡Apoyame para más proyectos! Este trabajo lo hice con mucho cariño y voy a ir mejorándolo mes a mes, así tienen una experiencia mucho mejor. Todo gratis, por el aprendizaje. Tu ayuda suma, y mucho, para mí. ¡¡Gracias!!

Invitar un cafecito (Mercado Pago, Argentina)

`cafecito.app/gastoncirvini`

Donar con PayPal (exterior)

`paypal.me/gastoncirvini`

ACERCA DEL AUTOR

Profesor de Grado Universitario en Historia, Magíster en Enseñanza en Escenarios Digitales, gamificador y programador.

CONTACTO Y OPINIONES

josegastoncirvini@gmail.com

© 2026 José Gastón Cirvini. Todos los derechos reservados. Uso libre y gratuito para aprender y enseñar. No se permite copiar, modificar, redistribuir ni vender este material sin autorización escrita del autor.